126 APRIL 1987

2.95 (3.95 CANADA)

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER

BUILDING BETTER BRAINS:
Artificial Neural Networks

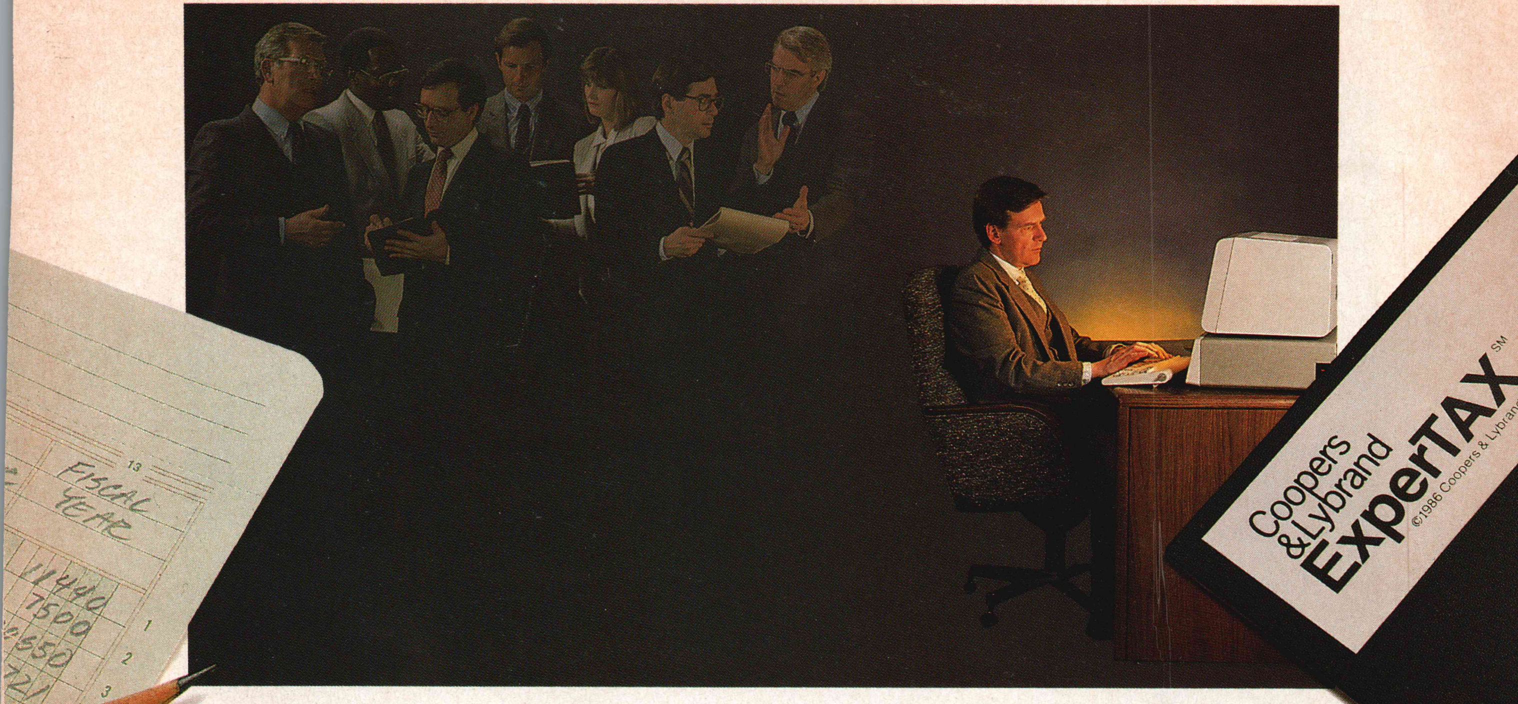
Four Mac PROLOGS Reviewed

Quick Expert Systems

Educating Programmers

Languages:
C Version of Nroff
Forth Programmers
Object Oriented LISPs
New BASIC Subroutines

0 38351 16562 8 04



Coopers & Lybrand selected Gold Hill for their expert tax system.

Expert systems like Coopers & Lybrand's ExperTAX℠ are powerful tools that can be employed for corporate clients in an increasingly sophisticated and complex business environment. Designed to assist staff accountants with tax accrual and tax planning issues, ExperTAX℠ was developed by Coopers & Lybrand with Gold Hill's **Golden Common LISP**, for use on an IBM PC XT, AT or compatible.

With ExperTAX, Coopers & Lybrand's partners and staff can provide corporate clients with the combined competence and experience of many of the firm's tax and audit professionals. Each time an individual client interacts with Coopers & Lybrand, it's as if that specialized tax knowledge is available in "computerized" form.

It's hard to believe that a sophisticated expert system like this one could be developed on a microcomputer. But it's no less amazing than the AI applications that Beckman Instruments, Honeywell, Martin Marietta, the USDA and other leading organizations are developing and delivering on IBM PC XTs and ATs, using Gold Hill.

That's why IBM, Digital and TI distribute **Golden Common LISP** on their PCs. And why **GCLISP** is already the accepted standard for AI on the PC.

Gold Hill also offers developers a family of tools for PC-based expert system development and delivery. **Golden Common LISP DEVELOPER**—including 286- and 386-based software—lets you create your own expert systems that will run on PCs. Add Gold Hill's **GCL RUN** and you can take AI applications developed on PCs—and deliver runtime versions of them for internal delivery or external distribution. You can even network these applications with leading AI hardware using **Golden Common LISP NETWORKS**. Plus, there's the **Gold Hill 386 LISP System**—a 386-based LISP system that transforms your existing PC into a LISP machine. Finally, for the most powerful expert system building environment soon to be available on the PC, there's Gold Hill's **GoldWorks**.

Whether your expert system application is in financial planning, or in something just as important to you, let Gold Hill help with AI on the PC. To learn more, call toll-free today:

1-800-242-LISP

GOLD HILL™

In Mass.: (617) 492-2071
Gold Hill Computers, Inc.
163 Harvard St.,
Cambridge, MA 02139

Gold Hill. The expert in AI on PCs.

Circle **no. 291** on reader service card.

PC MAGAZINE EDITOR'S CHOICE

*...one package stands out as the best support available for Turbo Pascal programmers: **Blaise Computing's Turbo Power Tools.** This definitive set of prewritten Pascal functions and procedures will make the life of any programmer—from the beginner to the hard-core professional—easier and more productive.*

Power Tools Plus™
Window Routines.
Memory Resident Routines.
Routinely.

ANOTHER PLUS FROM BLAISE COMPUTING

The best just got better! Turbo POWER TOOLS, acclaimed as the best programmer support package for Turbo Pascal, now has even more functions, more detailed documentation and more sample programs.

NO SECRETS

Turbo POWER TOOLS PLUS is crafted so that the source is efficient, readable and easy to modify. We don't keep secrets! We tell you exactly how windows are managed, how interrupt service routines can be written in Turbo Pascal, and how to write memory resident programs that can even access the disk. Maybe you've heard of some undocumented DOS features that resident programs use to weave their magic. Turbo POWER TOOLS PLUS documents these features and lets you make your own magic!

Here's just part of the PLUS in Turbo POWER TOOLS PLUS:

- **WINDOWS** that are stackable, removable, with optional borders and a cursor memory.
- **FAST DIRECT VIDEO ACCESS** for efficiency.
- **SCREEN HANDLING** including multiple monitor and EGA 43-line support.
- **POP-UP MENUS** which are flexible, efficient and easy to use, giving your applications that polished look.
- **INTERRUPT SERVICE ROUTINES** that can be written in Turbo Pascal without the need for assembly language or inline code.
- **INTERVENTION CODE** lets you develop memory resident applications that can take full advantage of DOS capabilities. With simple procedure calls, you can "schedule" a Turbo Pascal procedure to execute either when a "hot key" is pressed, or at a specified time.
- **PROGRAM CONTROL ROUTINES** allow you to run other programs from Turbo Pascal, and even execute DOS commands.
- **MEMORY MANAGEMENT** allows you to monitor, allocate and free DOS-controlled memory.
- **DIRECTORY AND FILE HANDLING** support to let you take advantage of the newer features of DOS including networking.
- **STRING** procedures allowing powerful translation and conversion capabilities.
- **FULL SOURCE CODE** for all included routines, sample programs and utilities.
- **DOCUMENTATION, TECHNICAL SUPPORT** and attention to detail that have distinguished Blaise Computing over the years.

Turbo POWER TOOLS PLUS** supports Turbo Pascal Version 2.0 and later and is just **$99.95.

*Another quality product from Blaise Computing: **Turbo ASYNCH PLUS™***

A new package which provides the crucial core of hardware interrupt support needed to build applications that communicate. ASYNCH PLUS offers simultaneous buffered input and output to both COM ports at speeds up to 9600 baud. The XON/XOFF protocol is supported. Now it also includes the "XMODEM" file-transfer protocol and support for Hayes compatible modems.

The underlying functions of Turbo ASYNCH PLUS are carefully crafted in assembler for efficiency and drive the UART and programmable interrupt controller chips directly. These functions, installed as a runtime resident system, require just 3.2K bytes. The high level function are all written in Turbo Pascal in the same style and format as Turbo POWER TOOLS PLUS. All source code is included for just $99.95.

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

ORDER TOLL-FREE 800-227-8087

Calif. residents call (415) 540-5441

YES, send me the PLUS I need! Enclosed is $______ for ☐ Turbo POWER TOOLS PLUS ☐ Turbo ASYNCH PLUS ☐ OTHER ______. (CA residents add 6½% Sales Tax. All domestic orders add $10.00 for Federal Express shipping.)

Name: ______ Phone: (___) ______

Shipping Address: ______

City: ______ State: ___ Zip: ______

VISA or MC #: ______ Exp. Date: ______

Circle **no. 159** on reader service card.

CONTENTS

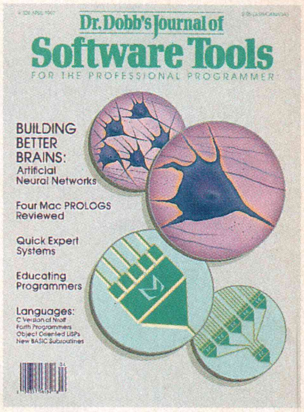
ARTICLES

REVIEWS

COLUMNS

FORUM

PROGRAMMER'S SERVICES



About the Cover

Neural network programming takes the human nervous system as its model for program structure. Better brains? No, but brainlike programs, yes.

This Issue

Our third annual Artificial Intelligence issue reflects the state of AI today. LISP is now common; PROLOG, which virtually didn't exist on micros in 1985, is everywhere; expert systems are just tools; and the cutting edge of AI work is the learning problem.

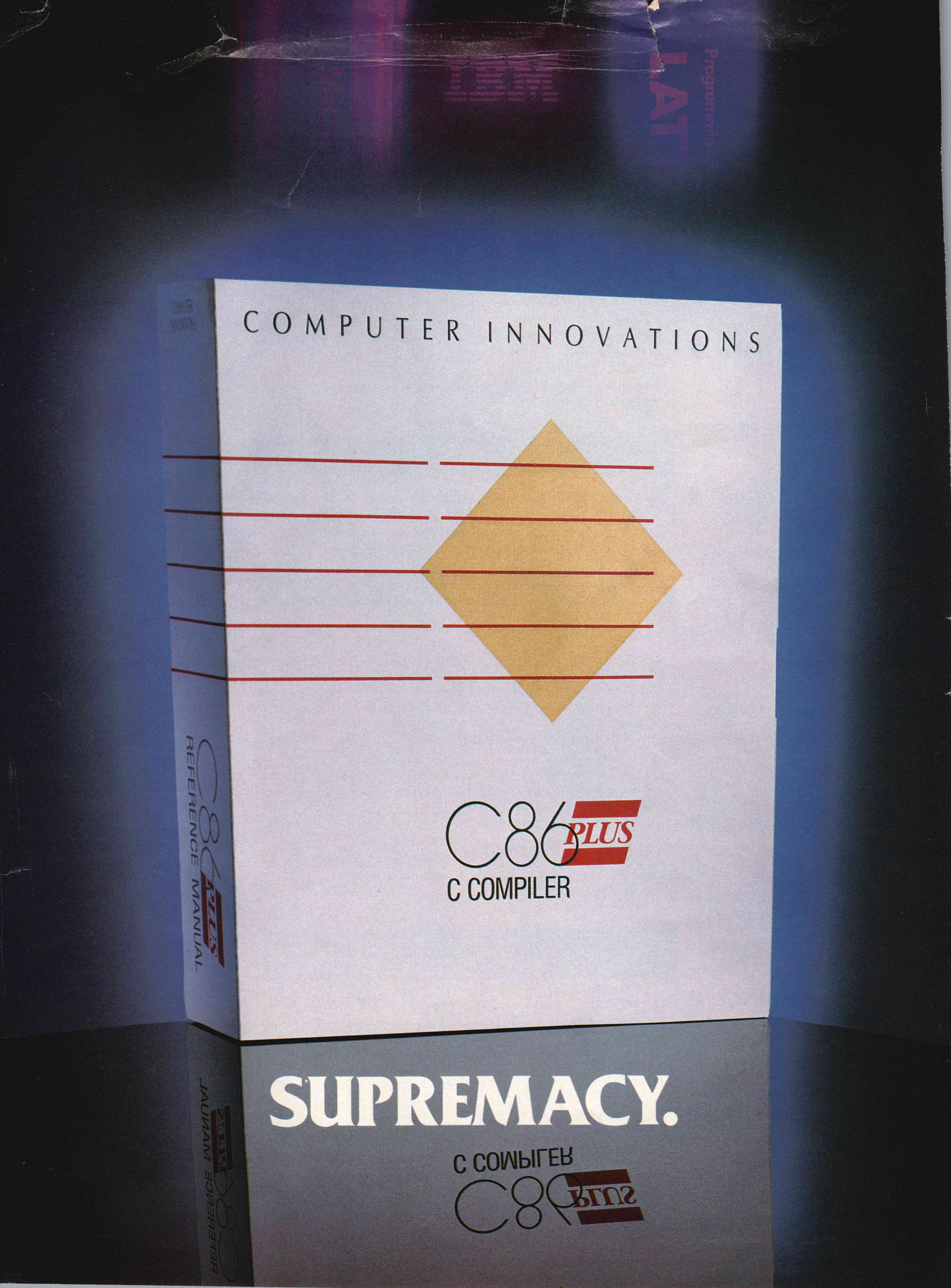
Next Issue

For much of its short lifespan (about 30 years), computer music has been primarily an ivory-tower pursuit—if you wanted to do serious work in the field of computer-generated music you looked to the university research labs. Today, music algorithms also originate in the homes and workshops of people like you—people who program music applications on computers such as the Amiga, Atari ST and Macintosh.

Our May feature article includes a brief history of computers in music and focuses on some recent developments in MIDI programming, sampling, transient-oriented synthesis methods, and programs that compose and collaborate on original music. We'll also have an article about how to design a software-based music recorder using MIDI.

bandwidth topic

entry point


COMPUTER INNOVATIONS
C86 PLUS
C COMPILER
C86 PLUS REFERENCE MANUAL
SUPREMACY.

SUBSTANTIATED.

SUPREMACY

It's a bold claim. A claim we're prepared to stake our reputation on. And at Computer Innovations, we've always taken our reputation very seriously.

It's no industry secret that the competitive C Compilers are at the end of their optimization cycle — they're just about as good as they are going to get. C86PLUS begins where everybody else has left off. It's an entirely new technology based on artificial intelligence and advanced compiler design techniques. Designed with the serious programmer in mind, C86PLUS provides the ultimate development environment, matching unparalleled execution speed with a host of productivity features.

FAST EXECUTION

- 20% faster than Microsoft C, version 4.0
- 70% faster than existing C86, version 2.3 (timings based on the classic sieve benchmark)

ANSI C COMPILER FEATURES

- Register variables
- Structure assignment
- Function prototypes
- New type modifiers
  - -near
  - -far
  - -signed
  - -const
  - -volatile
- Long double 80 bit floating-point
- Enumerator data types (enums)
- Extended preprocessor capabilities

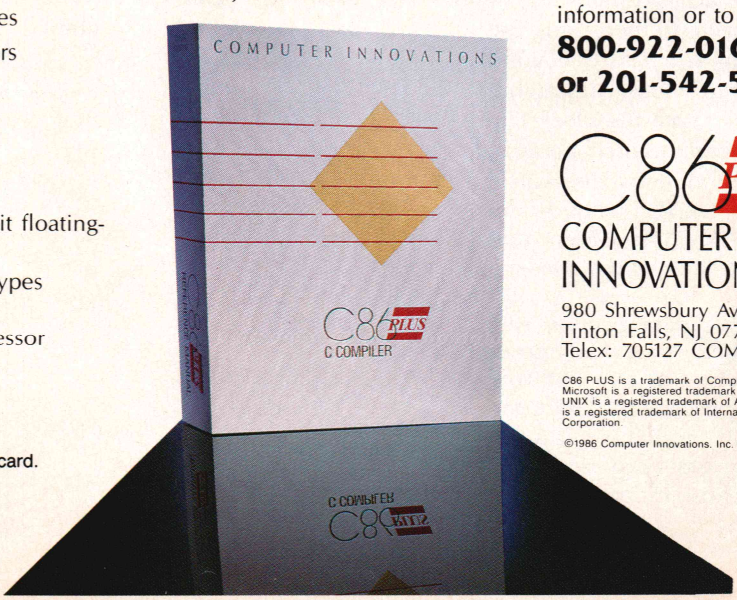
Circle **no. 96** on reader service card.

FULL CONTROL OVER COMPILATION ENVIRONMENT

- Small, Medium, and Large memory models
- 8086/80186 and 80286 code generation options
- In-line 8087/80287 floating point
- 8087/80287 auto detect emulator
- Source level debugger support
- Wild-card compilation
- Make utility
- ROMable code
- Linkable with macro assembler output
- Intel-standard OMF object files
- Optional assembly language output
- Warning level control

EXTENSIVE FUNCTION LIBRARIES FOR INCREASED PRODUCTIVITY

- Over 250 library functions
- Full ANSI C library
- Functional equivalents to most UNIX System V libraries
- Shared file and network support
- Low-level machine access functions
- IBM ROM BIOS support routines
- Fully compiled small, medium and large model libraries
- C library source code
- Run-time start-up source code
- Source code librarian
- Object code librarian



MICROSOFT COMPATIBILITY

If you're a current Microsoft user, we invite you to consider this simple point. C86PLUS will recompile most applications developed using MS-C without changes to your source code. You'll find that your application runs much faster.

PROVEN EXPERIENCE

In 1981, Computer Innovations and its founder, George Eberhardt, revolutionized the DOS programming world with the introduction of the first C Compiler for the PC called C86. Today, C86 boasts a satisfied and loyal user base of over 20,000 programmers worldwide. C86PLUS represents an extension of this expertise and reputation. It's backed with more than a decade of intensive research and development.

PROVEN SUPPORT

Making the claim that C86PLUS is supreme is one thing, standing behind it is another. Computer Innovations has always offered timely and intelligent technical support, and this is an important customer service which we do not intend to change.

CALL TO ORDER

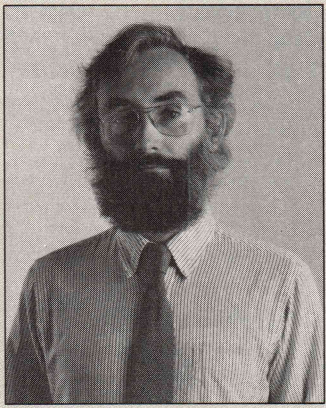
The call is on us. For more information or to order call:

800-922-0169
or 201-542-5920 (in NJ)

C86PLUS™

COMPUTER INNOVATIONS
980 Shrewsbury Ave.
Tinton Falls, NJ 07724, USA
Telex: 705127 COMP INNOV UD

EDITORIAL



Premise: Apple did several things right with the Mac 2.

First, note that the new generation of 32-bit personal computers will beckon to the computer - literate masses, not just to programmers and power users. As Bob Enyart wrote in the January 27 issue of *PC Week*, "the less sophisticated a user is, the more dependent on very high-powered I/O and intelligent user interfaces he will be." Performance will sell well.

Second, note that performance in the next generation machines is rapidly going to become bus-bandwidth-bound, particularly if those less-sophisticated users are demanding such memory-hungry features as high-powered I/O. (Color graphics and voice synthesis were among the features that Enyart alluded to.)

Don't take my word for it: "In the next few years, the bus bandwidth to the main memory is going to be *the* issue which determines performance."—Hal Hardenbergh, *The Programmer's Journal*, September/October 1986.

And in terms of the 68020 specifically, "memory access time is one of the major obstacles to increasing the performance of the 68020 beyond what it already provides. Although it is possible to increase the clock frequency, the performance benefit, unless accompanied by decreased system access time, will be of diminished value."—Doug MacGregor and Jon Rubinstein, *IEEE Micro*, December 1985.

The disk drive data-transfer rate will also be critical. It's already possible to see the crippling effect of slow drives on fast processors. The Compaq Deskpro 386, which is just a fast micro after all, works as well as it does partly because it doesn't have a slow disk drive.

Commercial disk drive technology has been a five-megabit-per-second world, and five megs will bore a 386 or 68K unmercifully. Two faster drive interfaces currently getting attention are the SCSI (Small Computer Systems Interface) and the ESDI (Enhanced Small Disk Interface). SCSI will do up to 12 megabits; ESDI will do up to 24 megabits. SCSI is smart and cheap (a very-low cost do-it-yourself SCSI drive for the Mac was presented in the September 1985 issue of *DDJ*) and is becoming something of a standard.

In the light of this, consider Apple's choices of Nubus and an SCSI disk interface: an open-architecture, relatively low-cost, relatively high-bandwidth bus and a relatively low-cost, relatively high-bandwidth disk interface. Power, dare I say, to the people.

Of course, they left the CPU burdened with driving the screen. But I only said they did *some* things right.

Michael Swaine

Michael Swaine
editor-in-chief

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Editorial
Editor-in-Chief *Michael Swaine*
Managing Editor *Vince Leone*
Assistant Editors *Sara Noah Ruddy*, *Levi Thomas*
Technical Editor *Allen Holub*
Consulting Editor *Nick Turner*
Contributing Editors *Ray Duncan*, *Michael Ham*, *Bela Lubkin*, *Namir Shammas*, *Ernest R. Tello*
Copy Editor *Rhoda Simmons*

Production
Production Manager *Bob Wynne*
Art Director *Michael Hollister*
Assoc. Art Director *Joe Sikoryak*
Typesetter *Jean Aring*
Technical Illustrator *Frank Pollifrone*
Cover Illustrators *Frank Pollifrone*, *Mark Schroeder*

Circulation
Circulation Director *Maureen Kaminski*
Newsstand Sales Mgr. *Stephanie Barber*
Book Marketing Mgr. *Jane Sharninghouse*
Circulation Coordinator *Kathleen Shay*

Administration
Finance Director *Kate Wheat*
Business Manager *Betty Trickett*
Accounts Payable Supv. *Mayda Lopez-Quintana*
Accts. Receivable Supv. *Laura Di Lazzaro*

Account Managers
Lisa Boudreau *(415) 366-3600*
Gary George *(404) 897-1923*
Michael Wiener *(415) 366-3600*
Cynthia Zuck *(718) 499-9333*
Promotions/Srvcs.Mgr. *Anna Kittleson*
Advertising Coordinator *Charles Shively*

M&T Publishing Inc.
Chairman of the Board *Otmar Weber*
Director *C.F. von Quadt*
President and Publisher *Laird Foshay*
Associate Publisher *Michael Swaine*

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly by M&T Publishing Inc., 501 Galveston Dr., Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points.

Article Submissions: Send manuscripts and disk (with article and listings) to the Editor.

***DDJ* on CompuServe:** Type GO DDJ

Address Correction Requested: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 27809, San Diego, CA 92128. **ISSN 0888-3076**

Customer Service: For subscription problems call: outside CA (800) 321-3333; in CA (619) 485-9623 or 566-6947. For book/software order problems call (415) 366-3600.

Subscriptions: $29.97 per 1 year; $56.97 for 2 years. Canada and Mexico add $27 per year airmail or $10 per year surface. All other countries add $27 per year airmail. Foreign subscriptions must be prepaid in U.S. funds drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service Inc., 386 Park Ave. South, New York, NY 10016; (212) 686-1520 TELEX: 620430 (WUI).

People's Computer Company
Dr. Dobb's Journal of Software Tools is published by M&T Publishing Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.

Our thanks to NASA for supplying this computer enhanced ultraviolet photo taken by Skylab IV of a solar prominence reaching out 350,000 miles above the sun's surface

Genius Begins With A Great Idea ...

But The Idea Is Just The Beginning

What follows is the time consuming task of giving form and function to the idea.

That's why we concentrate on building into our software development systems functions and features that help you develop your software ideas in less time and with less effort.

We've started 1987 by releasing new versions of our MS-DOS, Macintosh, Amiga, ROM, and Apple // C development systems. Each system is packed with new features, impressive performance, and a little bit more genius.

Aztec C86 4.1
New PC/MS-DOS • CP/M-86 • ROM

Superior performance, a powerful new array of features and utilities, and pricing that is unmatched make the new Aztec C86 the first choice of serious software developers.

Aztec C86-p Professional System $199
• optimized C with near, far, huge, small, and large memory + Inline assembler + Inline 8087/80287 + ANSI support + Fast Float (32 bit) + optimization options • Manx Aztec 8086/80x86 macro assembler • Aztec overlay linker (large/small model) • source level debugger • object librarian • 3.x file sharing & locking • comprehensive libraries of UNIX, DOS, Screen, Graphics, and special run time routines.

Aztec C86-d Developer System $299
• includes all of Aztec C86-p • Unix utilities make, diff, grep • vi editor • 6 + memory models • Profiler.

Aztec C86-c Commercial System. $499
• includes all of Aztec C86-d • Source for library routines • ROM Support • CP/M-86 support • One year of updates.

Aztec C86 Third Party Software

A large array of support software is available for Aztec C86. Call or write for information. The following is a list of the most requested products: Essential Graphics • C Essentials • C Utility Library • Greenleaf Com. • Greenleaf General • Halo • Panel • PC-lint • PforCe • Pre-C • Windows for C • Windows for Data C terp • db Vista • Phact • Plink86Plus • C-tree.

CP/M • TRS-80 • 8080/Z80 ROM

C compiler, 8080/Z80 assembler, linker, librarian, UNIX libraries, and specialized utilities.

Aztec C II-c (CP/M-80 & ROM). $349
Aztec CII-d (CP/M-80) $199
Aztec C80 (TRS-80 3&4) $199

Aztec C68k/Am 3.4
New Amiga Release

Amiga user groups across the USA voted Aztec C68k/Am release 3.3 the best Software Development System for the Amiga. Release 3.4 is more impressive.

Aztec C68k/Am-p Professional $199
A price/feature/performance miracle. System includes: optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Amiga libraries • examples.

Aztec C68k/Am-d Developer $299
The best of Manx, Amiga, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep and vi.

Aztec C68k/Am-c Commercial $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

Aztec C68k/Mac 3.4
New Macintosh Release

For code quality, reliability, and solid professional features, Aztec C for the Macintosh is unbeatable. This new release includes features and functions not found in any other Macintosh C development system.

Aztec C68k/Mac-p Professional $199
• optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Macintosh libraries • examples.

Aztec C68k/Mac-d Developer. $299
The best of Manx, Macintosh, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep • vi editor.

Aztec C68k/Mac-c Commercial $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

Aztec C65
New ProDOS Release

Aztec C65 is the only commercial quality C compiler for the Apple II. Aztec C65 includes C compiler, 6502/65C02 assembler, linker, library utility, UNIX libraries, special purpose libraries, shell development environment, and more. An impressive system.

Aztec C65-c Commercial $299
• runs under ProDOS • code for ProDOS or DOS 3.3

Aztec C65-d Developer $199
• runs under DOS 3.3 • code for DOS 3.3

Aztec ROM Systems
6502/65C02 • 8080/Z80 • 8086/80x86 • 680x0

An IBM or Macintosh is not only a less expensive way to develop ROM code, it's better. Targets include the 6502/65C02, 8080/Z80, 8086/80x86, and 680x0.

Aztec C has an excellent reputation for producing compact high performance code. Our systems for under $1,000 outperform systems priced at over $10,000.

Initial Host Plus Target $750
Additional Targets $500
ROM Support Package $500

Vax, Sun, PDP-11 ROM HOSTS

Call for information on Vax, PDP-11, Sun and other host environments.

C' Prime
PC/MS-DOS • Macintosh
Apple II • TRS-80 • CP/M

These C development systems are unbeatable for the price. They are earlier versions of Aztec C that originally sold for as much as $500. Each system includes C compiler, assembler, linker, librarian, UNIX routines, and more. Special discounts are available for use as course material.

C' Prime $75

Aztec Cross Development Systems

Most Aztec C systems are available as cross development systems. Hosts include: PC/MS-DOS, Macintosh, CP/M, Vax, PDP-11, Sun, and others. Call for information and pricing.

How To Become An Aztec C User

To become a user call 800-221-0440. From NJ or international locations call 201-542-2121. Telex: 4995812 or FAX: 201-542-8386. C.O.D., VISA, MasterCard, American Express, wire (domestic and international), and terms are available. One and two day delivery available for all domestic and most international destinations.

Aztec C is available directly from Manx and from technically oriented computer and software stores. Aztec Systems bought directly from Manx have a 30 day satisfaction guarantee.

Most systems are upgradable by paying the difference in price plus $10. Site licenses, OEM, educational, and multiple copy discounts are available.

MANX

Circle 108 on reader service card.

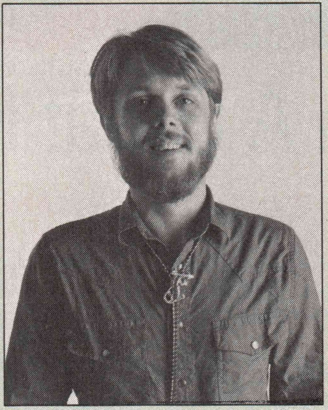
Manx Software Systems
1 Industrial Way, Eatontown, NJ 07724

To order or for more information call today.

1-800-221-0440

In NJ or international call (201) 542-2121 • TELEX: 4995812

RUNNING LIGHT



Like many of you, I'm a habitual hacker. But, also like most of you, I get burned out from time to time. When I joined *DDJ* last February, I was more than ready to stop programming for a while. I had just finished working on a large Macintosh project for a company that promptly went out of business. This kind of thing happens to freelance programmers once in a while, I fear. You put a lot of sweat, time, and craftsmanship into a project that ends up never seeing the light of day. And, often as not, you end up never seeing the paycheck for all that wasted effort either. These are the events that lead to programmer burnout, and I had it bad. When the opportunity to edit *DDJ* came along, I decided to take a break from programming and explore the world of magazine editing . . . frankly, I needed a change of pace; at *DDJ* I could talk and write about programming and get paid for it! An exciting possibility to be sure.

Now, after almost a year of talking about programming and reading about other people's programming projects, I'm itchy to get back into the thick of things again. So, starting next month, I'll be stepping aside as editor and taking on the role of consulting editor. This way I get more time to program and still keep my hand in around here. (Sometimes you *can* have your cake and eat it too!) You will also hear from me from time to time in the Right to Assemble column.

In the meantime, this is our annual AI issue, with a feature article on a program that can be taught to recognize ASCII characters in graphic form. SILOAM represents the beginnings of the new field I mentioned recently—neural networks. There was a brief flurry of interest in such networks several years back, with the invention of various simple neural circuits that could be simulated relatively easily. But it amounted to little, partly because of limited hardware and partly because the models were not sophisticated enough. Now the field is heating up again, especially as we gain the power to simulate somewhat larger nets. As is typical of new fields, theories abound, and there's a lot of work to be done. We'd like to follow the developments closely; if you're working in the field, why not give us a call or send a letter?

In upcoming issues we'll be resurrecting the Professional Programmer department in our Programmer's Services section. The department will deal with various topics pertinent to those of you who are making your living as programmers—topics such as software copyright laws, products for producing firmware, and how to get your software published. We'll also have interviews with some well-known programmers and hear about their projects and their preferred tools for software development. If you have suggestions about what you'd like to see in the Pro Pro department, send a note to our assistant editor Levi Thomas. You can also reach Levi on CompuServe—ID number 76703,4060.

Who knows, we may even write something about how to avoid programmer burnout—if we can find someone who knows the trick to it. . . .

It's been a heck of a lot of fun, folks. Stay tuned!

Nick Turner
editor

ARCHIVES

The Expanding Mag

"Once again, we've published our largest issue ever! Our expansion is exciting; but at the same time, we are conscious that bigger is not always better. In this case, however, our increase in size will allow us options we haven't had in the past. With more editorial pages, we have the opportunity to present more variety each month, even when we run larger pieces, as we have in this issue and in the last."—*editorial, Reynold Wiggins,* DDJ, *August 1983.*

Predictions in AI

"The biggest revolution will come in terms of software. 'Actor-based' languages such as Smalltalk will allow programs to alter themselves to the user's wishes. If various present-day languages were embedded in such an actor language, ten strange new customized languages for specific purposes could be generated on demand, merely by the user having a 'conversation' with the larger host AI/actor program."—*"5th Generation Computers," Richard Grigonis,* DDJ, *December 1982.*

"So what have the Japanese selected as the lingua franca of their AI research? Not LISP, but PROLOG, an obscure language developed by the French and 'polished' by the British. PROLOG is the fundamental error in the otherwise sound Japanese Fifth Generation Project."—*"And Still More Fifth Generation Computers," Richard Grigonis,* DDJ, *August 1983.*

Ten Years Ago in *DDJ*

"We (the Professional Users Group) are not particularly interested in organizational clap-trap but rather in dissemination of both hard data and innovative fantasy, regarding technical features of small computer systems, both hardware and software. As an example of innovative fantasy, let me tell you about Dick Maus's project: to regard the game of 'life' as a subset of what Dick calls 'Superlife.' This algorithm will treat growth and decline phenomena in cell arrays. He sees applications for his simulation studies in such diverse fields as auto traffic flow, interaction of arrays of businesses in a given market, wave phenomena analysis, and nerve impulse transmission in neural tissue, just for starters."—*William J. Schenker, M.D., letter to* DDJ, *April 1977.*

. . . and a more-than-usually confused potpourri of interesting tidbits."—*last entry in table of contents,* DDJ, *April 1977.*

Dr. Dobb's Journal of Computer Calisthenics & Orthodontia
Running Light Without Overbyte

An Introduction to Wendin's Operating System Toolbox

This toolbox gives you a hands-on demonstration of how multitasking operating systems work.

Have you ever spent creative hours building things with Tinkertoys or an Erector set? Wendin's Operating System Toolbox gives you the same hands-on experience with operating system architecture that an Erector set gives you with mechanical engineering. You can use the toolbox as a development tool to write a custom operating system, or as a learning aid to discover how multitasking operating systems work. If you've ever wanted to build a better DOS, or see how big operating systems like VAX/ VMS and UNIX really work, this product is for you.

The toolbox comes with complete source code, ready to compile. The roughly 15,000 lines of source code are libraried to make the system fit on two floppy disks. About 85% of the system is written in C, and the remainder in assembly language for speed. The package comes with a 300+ page manual that describes how to build systems with the toolbox, and how the kernel works.

Kernel Architecture

The basic architecture of the kernel was designed around the VAX/VMS operating system. As shown in Figure 1, there are three major subsystems: the scheduler, memory manager, and I/O system. The kernel's subsystems communicate through common data structures which describe processes, memory usage, and devices.

Processes (or tasks) execute in 4 different access modes, depending on the level of access to system data structures they need. USER access mode, the least privileged, is where application programs execute, to keep other users' data protected.

> *"I heartily recommend the OST as an educational tool for experienced programmers of all types."*
> **Jason Levitt**
> *March Software Reviews for Byte Magazine*

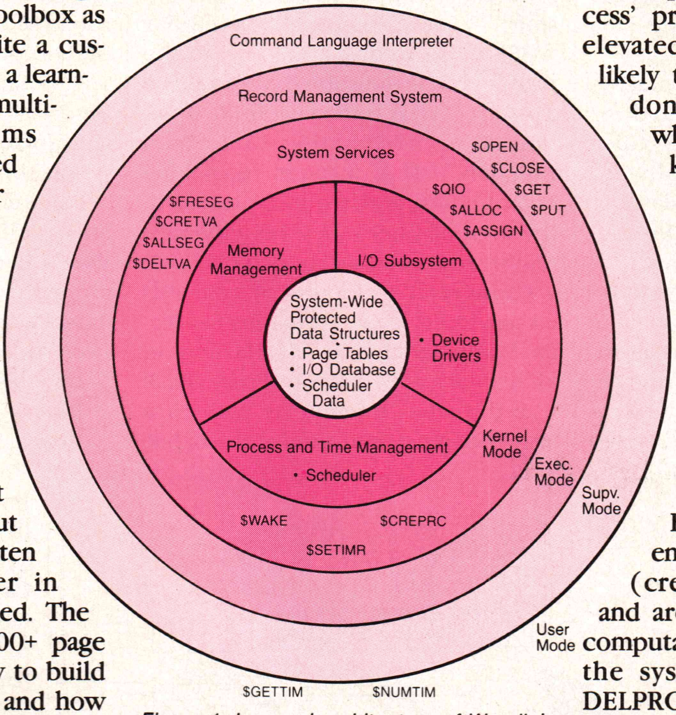


Figure 1. Layered architecture of Wendin's Operating System Toolbox.

The command language interpreter, or user interface, executes in SUPERVISOR mode, so that it can oversee the execution of user programs. The built-in Record Management System (RMS) executes in EXECUTIVE mode. Device drivers and process control services execute in KERNEL mode, the most privileged access mode. This allows other layers to make calls to a device specific layer, thereby making them device independent.

Scheduling

The toolbox schedules processes using the same sophisticated algorithm that VAX/VMS uses. Ready processes are collected in *computable queues* according to their current priority (0-31). Waiting processes are collected in other queues depending on their *state* (hibernating, suspended, waiting for I/O to complete, and so forth). A process' priority may be temporarily elevated so that it will be more likely to execute first when it is done waiting. For example, when a process is waiting for keyboard input, it remains in LEF (an I/O wait state) until the read has completed. When the read completes, the process priority is elevated to make response time quick.

Processes are moved by the scheduler from state to state according to the diagram shown in Figure 2. New processes enter the system via a CREPRC (create process) system call, and are immediately placed on a computable queue. Processes leave the system when deleted by a DELPRC (delete process) system call, which automatically makes the process current before deleting it. This means that there is only one path in or out of the system.

To select the next current process, the scheduler scans the computable queues for the highest priority process, and removes it from its queue. The process is then placed in the CUR state, and is executed. When a system event such as a timer tick or keypress occurs, the current process is placed back on a computable queue, and the event is serviced by some kernel component. After servicing the event, the scheduler finds another computable process to run.

When a process requests to be placed in a wait state such as

LETTERS

Software Gap

Dear *DDJ*,

I'm responding to the two "name witheld" letters in the January 1986 issue.

To the person whose managers have business backgrounds and don't understand software: I'd say turnabout is fair play. You know software and have strongly held standards—witness your letter and your *DDJ* subscription. You were interviewed for your current job and were deemed acceptable. In an interview, you have rights, too. Interview the person interviewing you. No need to be overly aggressive about it—there's always a line to walk—but see if you have common ground. Without that common ground, you're going to be unhappy. A good manager will recognize this, and you don't want to work for a bad one.

To the person concerned about sloppy code: You mentioned "it is sometimes difficult to determine when to stop coding and start stubbing." True enough, unless you have a plan. Let me suggest one. I like to add code to a running program. It's more enjoyable because the feedback is right there—it shows on the screen, the device runs, whatever. Debugging is simpler—you can see. Feedback is the crucial element in debugging.

So my suggestion: Write the main routine and enough I/O to get some feedback. Stub everything else. Get that running. So it's too easy/simple—who cares? You have a place to stand, and the CPU is your lever. Now add code that makes the program do something simple. Then something else. Keep that feedback there. Otherwise the job can become work.

I think it's neat that I can go somewhere five days a week to have fun and actually get paid for it.

Joe Osgood
14930 Hartland St.
Van Nuys, CA 91405

More Searching for a Sine

Dear *DDJ*,

Richard A. Campbell's use of a Taylor's series approximation for the sine ["In Search of a Sine," December 1986] is the wrong approach to the problem. The question that should be asked is "What is the best finite polynomial approximation to a given function?" The answers have long been known and are to be found in the area of Chebyshev polynomials; Mainframe programmers have done a considerable amount of research on this topic. *Approximations for Digital Computers* by Cecil H. Hastings et al. (Princeton University Press, 1955) lists the sine function as well as a wide variety of common and esoteric functions as part of a research study by the RAND Corp.

If you consider a poynomial approximation of three terms for the sine, the values of C1 = 1.5706268, C2 = −0.6432292, and C3 = 0.0727102 are derived from approximating theory, whereas Campbell's Taylor's series coefficients are C1 = 1.570795, C2 = −0.645921, and C3 = 0.07948765. For a polynomial approximation of four terms, the coefficients are the same for the approximating theory and the finite Taylor's series expansion.

Hopefully this will help Campbell and others in their investigations in this area. Too many implementations of computer languages ignore the vast amount of work that has gone before. The implementation of the sine and other well-known functions points this out.

I should also point out that Radio Shack offers a BASIC program that yields 15-digit accuracy on a 4K machine for a wide variety of functions (Level II Double-Precision Subroutine Program, catalog number 26-1704).

Douglas Ingalls
Ithaca College
Ithaca, New York 14850

"Okay—I understand that it can see the difference between a cat and a dog—but does it know if their intentions are hostile?"

Dear *DDJ*,

In December 1986 you ran an article in which Richard Campbell recounted his difficulty in locating a suitable approximation for the sine function.

Maybe I can make his day. I faced a similar problem a few years ago, and in my searches I came across a book that consists almost entirely of approximations—polynomial approximations, recursive approximations (!), infinite series, and infinite products—not to mention all the useful identities, integrals, and derivatives thereof. Would you believe a trig table with values for the sine and cosine functions to 23 significant places? (Great for checking up on that library approximation you're using!) And more.

I'm talking about the *Hand-*

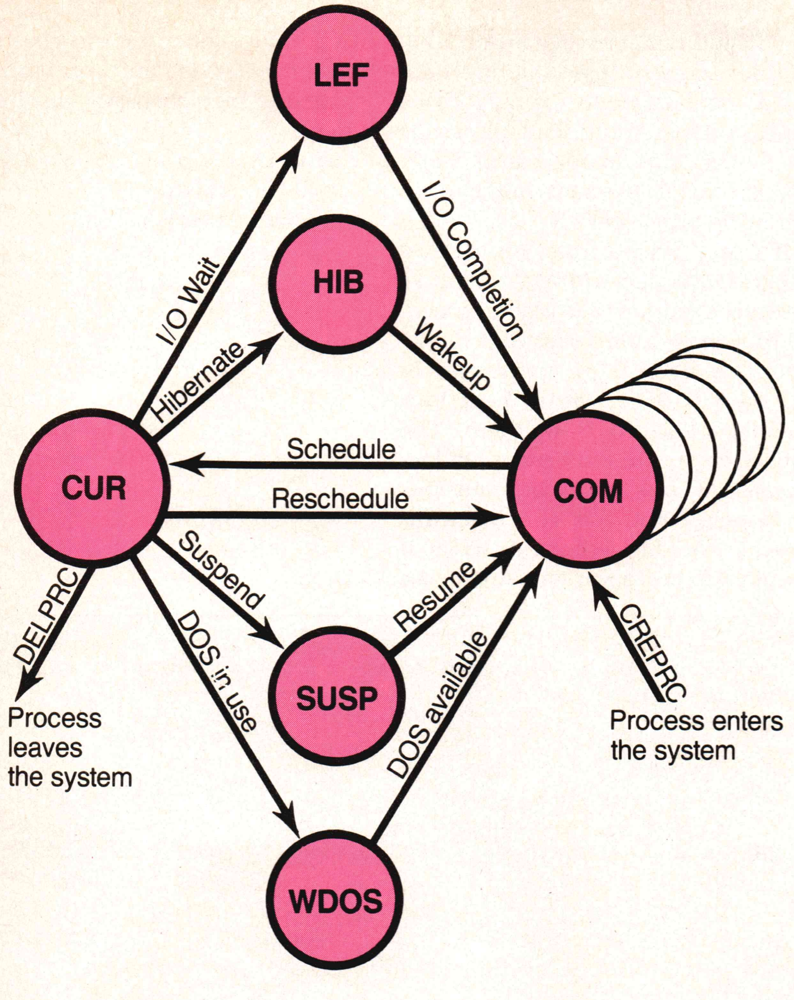


Figure 2. Scheduling paths and process states.

HIBernate or SUSPend, or if it requests a system resource that is not currently available such as a waited I/O or a DOS service, it is moved from the CUR state to the appropriate waiting queue (LEF, HIB, SUSP, or WDOS) until the resource becomes available. Once the resource is made available, the process is moved from its wait state to a computable queue, where it may be selected by the scheduler. In the case of HIB and SUSP, the process will wait until another process makes it computable with a WAKE or RESUME system call. A process can also wake itself up with a scheduled wakeup by calling the SCHDWK system call.

Memory management

The memory manager keeps track of available memory in 512-byte chunks called pages just like VAX/VMS. It uses a table of page descriptors to keep track of which processes own the individual pages, and which access mode they were allocated in. Pages can be assigned to a specific process through the CRETVA (create virtual address) system call, and returned to the system with the DELTVA (delete virtual address) system call. Pages can also be allocated for global use and assigned a name so that processes can share data in memory. A process that wants to allocate memory to share with other processes calls the CREGBL (create global memory block) system call. Other processes can then access the global memory block by name using the ACCGBL (access global memory block) system service. When a global memory block is no longer needed, it may be returned to the system with a DELGBL (delete global memory block) system call.

Input/Output System

The I/O system handles all process requests for interaction with hardware and software devices. Hardware devices represent actual hardware, such as disks, terminals, and printers. Software devices include mailboxes (memory-resident data pipes) and the null device (bit bucket). I/O is supported through two levels of organization: device-specific (QIO) and device-independent (RMS). QIO device drivers deal directly with the physical properties of the device (for example, track size on a disk, or baud rate for a terminal). The RMS (Record Management System) level supports device-independent system calls which call the QIO level to do the work. This means that RMS isn't responsible for knowing about the specific characteristics of a device.

The User Interface

The final and most important component of an operating system built with the toolbox is the user interface. This C function is called by the kernel whenever it makes a new process. It is up to the designer to determine what a process should do. For example, some processes should read commands from their standard input device. Others should just run a program and leave the system. The user interface can be as simple as the MS-DOS interface or as sophisticated as the Macintosh windowing environment. As a development kit, it's powerful.

As a learning aid, it's fascinating. Wendin's Operating System Toolbox is an in-depth tour of an interesting field that simply can't be passed up for the price.

Minimum system requirements include an IBM PC/XT, AT, or true compatible with at least 256K running MS-DOS or PC-DOS.

Operating System Toolbox: $99
Wendin, Inc.
P.O. Box 3888
Spokane, WA 99220
(509) 624-8088

(continued from page 10)

book of Mathematical Functions, edited by Milton Abromowitz and Irene A. Stegun, New York: Dover, 1965, originally published by the National Bureau of Standards of all people. After looking at dozens and dozens of mathematics and computer handbooks, searching in vain for an approximation of the gamma function, I finally came across this gem and had a rapturous religious experience on the spot. Now my copy resides next to Knuth, K & R, and other sacred writings of computer lore. I don't know how many times it has saved my derrière.

Your local college library almost certainly has a copy—go take a look, and see if it isn't everything I say it is.

William Zeitler
9 Pajaro Way
Salinas, CA 93901

Feedback

Dear *DDJ*,

You asked for input in the January issue, so here it is. First the complaint. At the bottom of page 16, you state: "you can move up from the 68008, which is roughly comparable to a fast Z80 in power." That is a gross misrepresentation—the 68008 is about four times faster than the 6809, and the 6809 runs rings around the Z80 (see the benchmarks published in *Interface Age*, April 1983). I agree that there have been several poor implementations of the 68008 (Hazelwood) that ran very slowly; however, a good hardware design (Peripheral Technology) will really fly.

In some instances the 68008 is faster than the 68000. A hard-disk driver where the sector buffer must be read and placed in memory is one such instance (see Example 1, right).

In the Right to Assemble column, the numbers project is super, and I am sure it will fill a large void in computer science literature. I wrote a 56-digit calculator for the 6809 in assembly language, and I could not find much help in any magazines or books. I had the same decision to make on the transcendental functions. The precision on the polynomial approximation was at the most 10 digits and I needed 56, so I had to use the series expansion. The arctan series would not converge around 1.5, so I devised a simple solution—I applied the half-angle formula three times and then multiplied the results by 8. This was so successful that I needed only 24 elements in the series for 56 digits of accuracy.

If I may brag a little, my CALC56 program calculates Pi (4*ATN(1)) in 14 seconds, accurate to 55 places. I needed four expansion series (sine, exponent of e, natural log, and arctan) in my calculator; all other functions could be derived from them. The source code is in 6809 assembly language and could easily be transported to the 68000. If you want to see this code, I would be willing to share. It would also be very interesting to see if you have better algorithms than I have. If you need any constants calculated to high accuracy, just let me know.

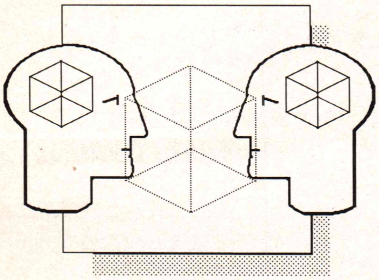
Dan Farnsworth
3646 Lantana Rd.
Lantana, FL 33462-2299

DDJ

```
In 68000 code you would do this (A1 points to the
destination.):

       MOVE.W #255,D1        counter                8
       LEA DCA,AO            DC addr               12
LOOP   MOVE.B (AO),(A1)+     move byte    12
       DBF D1,LOOP           loop         14     6656
       RTS                                         16
                                                 6692 §

In the 68008 you make the hardware so that the disk con-
troller addresses occupy 4 bytes consecutively:

       MOVEQ1.L #63,D1       counter                8
       LEA DCA,AO            DC addr               24
LOOP   MOVE.L (AO),(A1)+     move     long
                             word         40
       DBF D1, LOOP          loop         26     4158
       RTS                                         32
                                                 4222 §

If memory is available you can do this:

       LEA DCA, AO           DC addr               24
       MOVE.L (AO),(A1)+     move long             40
       MOVE.L (AO),(A1)+     move long             40
            !
            !
            !
       MOVE.L (AO),(A1)+     64 2 byte
                             instructions          40
                                                 2616
```

Example 1: *Sometimes the 68008 is faster than the 68000.*

VIEWPOINT



Education and Programming

There is a popular misconception (popular among educators and laymen alike) that programming and mathematics are somehow related. As a consequence an undue amount of math is forced down the gullets of unwilling undergraduates in every computer science program in the country. I feel that not only is this needless torture but it also can be counterproductive.

To my mind, a good, traditional, liberal-arts education—one that includes a little math but that also includes things such as English composition, history, and Latin—is better preparation for a programming career than a math- or science-oriented education. Many of the worst programs I've seen have been written by mathematicians and physicists and some of the best have been written by people with degrees in English and Russian Literature. How is this possible if programming is as grounded in math as some would have us believe? More to the point, many potentially excellent programmers are forced out of computer science programs every year because they don't have an aptitude for mathematics. The other side of the problem is that many programmers who do graduate are not fully in touch with the world around them because of the unwarranted dominance of technical training to the detriment of nontechnical but equally valuable kinds of knowledge.

by Allen Holub

Before continuing, I need to clarify my terms. There's a difference between *programming* and *computer science*, between the writing of programs and the study of them. Computer science has a great deal to do with mathematics; programming does not. I'm also differentiating between engineering and programming. You obviously need to understand mathematics to write a program that does mathematical things.

On the other hand, the creation of a computer program, regardless of the function of that program, is a different process from designing the math-related algorithms that the program implements. In any event, most programs don't implement mathematical algorithms, and if they do, the algorithms are often designed by someone other than the programmer. How much calculus is there in a word processor or a database application?

The skills you need to solve a math problem are virtually useless when it comes to writing a program. At the undergraduate level, at least, you solve a math problem by repetitively applying a set of memorized rules to an equation until one or more of those rules does something useful. If you tried to write a program this way, you'd never get beyond the *main( )* module. On the other hand, the process of writing an essay is almost identical to that of writing a program. You start both with an outline of some sort (what is a structure chart or a Warnier-Orr diagram if not an outline?), you have to organize both in the same way (a topic paragraph is a *main( )* subroutine, the routines that *main( )* calls are sections in the essay, and so forth), and the stepwise refinement of a program mirrors the development of an idea within an essay.

The study of a language, especially a classical language such as Latin, is also useful to programmers. It's not the Latin itself that's important—though a knowledge of Latin certainly does help improve your English—but rather it's the tools that you need to learn Latin that are important. You are really teaching yourself how to understand a large and complex system, and once you've understood how to learn the language, you can apply these same techniques to any complex system, such as a computer program.

These problems can be solved, to some extent, by a restructuring of the educational process. I'm arguing here, not for the restructuring of computer science programs as such, but rather for the creation of a new academic discipline entirely—one that's concerned primarily with programming. A little math is obviously required—basic algebra, a little Boolean and linear algebra, some set theory, and (a very little) calculus are all that's needed, however. These topics could be covered quite adequately in a well-designed one-semester course, and as I said earlier, a little math is just as much a part of a liberal-arts education as is English. If students were required to take a full year of English composition rather than calculus, not only would we have better-written programs, but we'd have readable documentation for them as well. The study of computer science as such could then be moved to the graduate level, as is fitting for an essentially academic discipline. A good parallel is the way that you'd earn a medical degree—a bachelor's degree in a related but more general field is required before you can enter a graduate-level M.D. program. Similarly, a degree in programming would be a prerequisite for a degree in computer science.

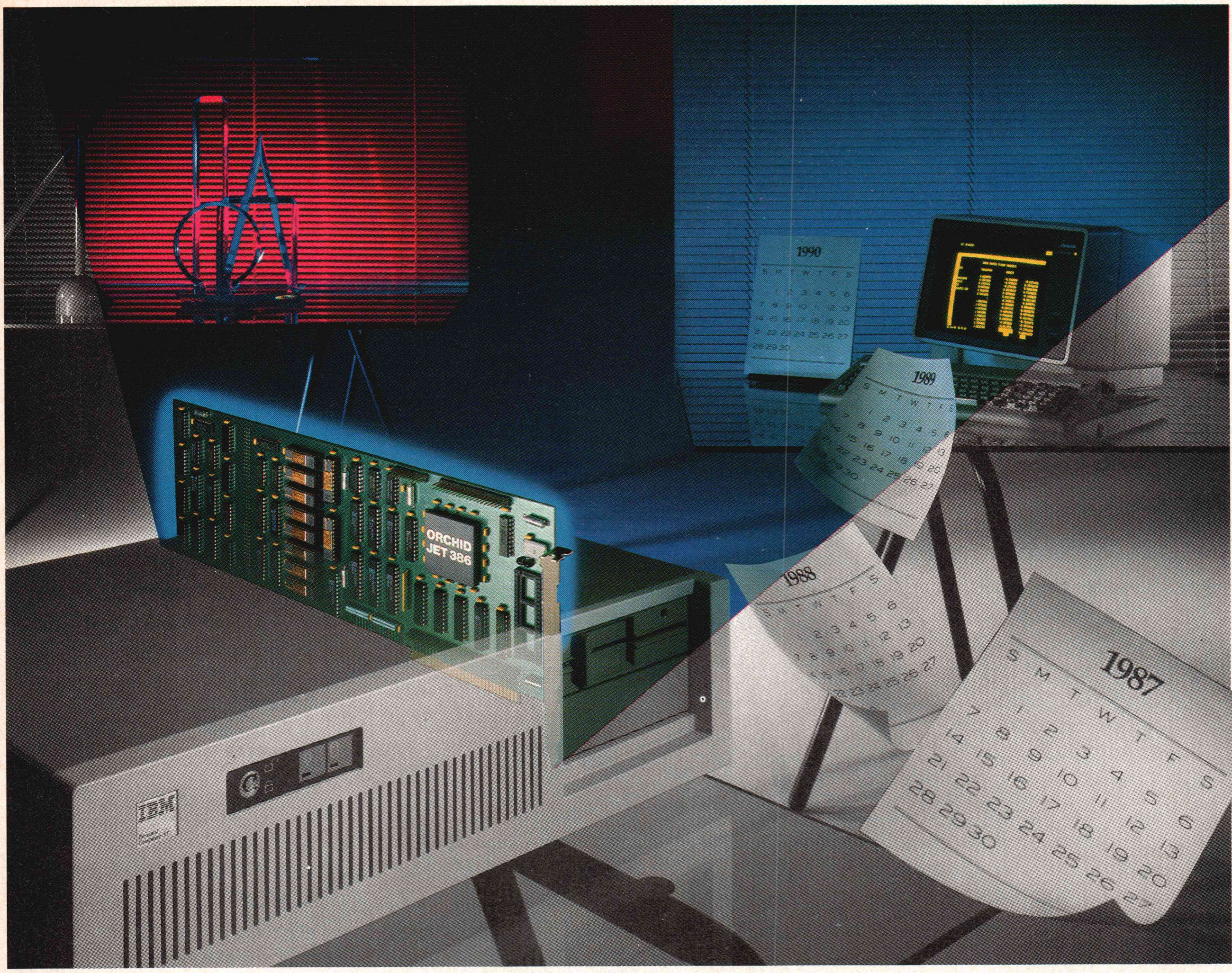
In the long run this sort of restructuring would give us both better programmers and better computer scientists.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 1.**

SPECIAL PRICE $1199.00

ORCHID'S JET 386™: POWER FOR THE FUTURE NOW



Jet 386 is the Ultimate Accelerator Upgrade for Your AT

Announcing an end to obsolescence. Orchid Technology's Jet 386™ accelerator card extends the life of your computer investment into the 1990s—it puts power in your AT that you won't outgrow.

Three Times Faster than an AT

It's up to three times faster than an AT depending on the application, and speed is just one benefit. Unequalled compatibility and provisions for upcoming 386 software mean your Jet 386 will handle whatever the future has in store: CAD, spreadsheets, networking...

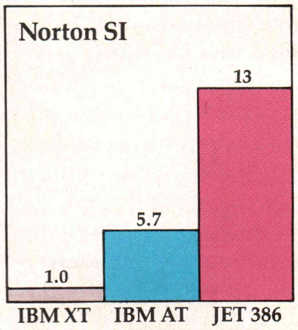
Easy Upgrade

Easy to use, there's nothing new to learn and no new programs to buy. At 25% of the cost of buying a new 386 PC, it's easy on your pocketbook, too.

From the People Who Started It All

Orchid combined 80386 power with the technology perfected for the XT in the TinyTurbo and PCturbo 286e. Like these critically acclaimed accelerators, Jet 386 is built for lasting value.

Call Orchid to find out how you can experience the future today. And ask how Orchid can modernize your whole office with turbos, graphics, networking, and multifunction products.



ORCHID
Innovative Add-Ons

45365 Northport Loop West
Fremont, CA 94538
415/490-8586 Tlx: 709289

Circle Reader Inquiry Number 130

An Artificial Neural Network Experiment

by Robert Jay Brown

This article describes an experimental computer program that could serve as one component of a computer vision system. The program simulates an artificial neural network and acts as a learning machine. It's implemented as a committee network of threshold logic units that functions as a trainable pattern recognizer for visual images composed of a rectangular array of pixels. Each pixel contains a single number representing the gray-scale value of that region of the field of view.

This program acts as a learning machine.

A robot vision system consists of many components:

- image capture, in which the image is converted from light or other radiation to an analog electrical signal
- image digitization and signal processing, in which the image is divided into a set of picture elements, or pixels, each of which is assigned a value representing the brightness and/or color of that part of the image
- region, edge, and boundary detection, in which the distinct visual elements of the image are separated
- image scaling and alignment, in which the digitized image is rotated, translated, and otherwise transformed to place it into some standard position and size
- image recognition, in which the preprocessed image is submitted to a pattern recognition algorithm to allow the machine to recognize what the image represents.

This article is about image recognition.

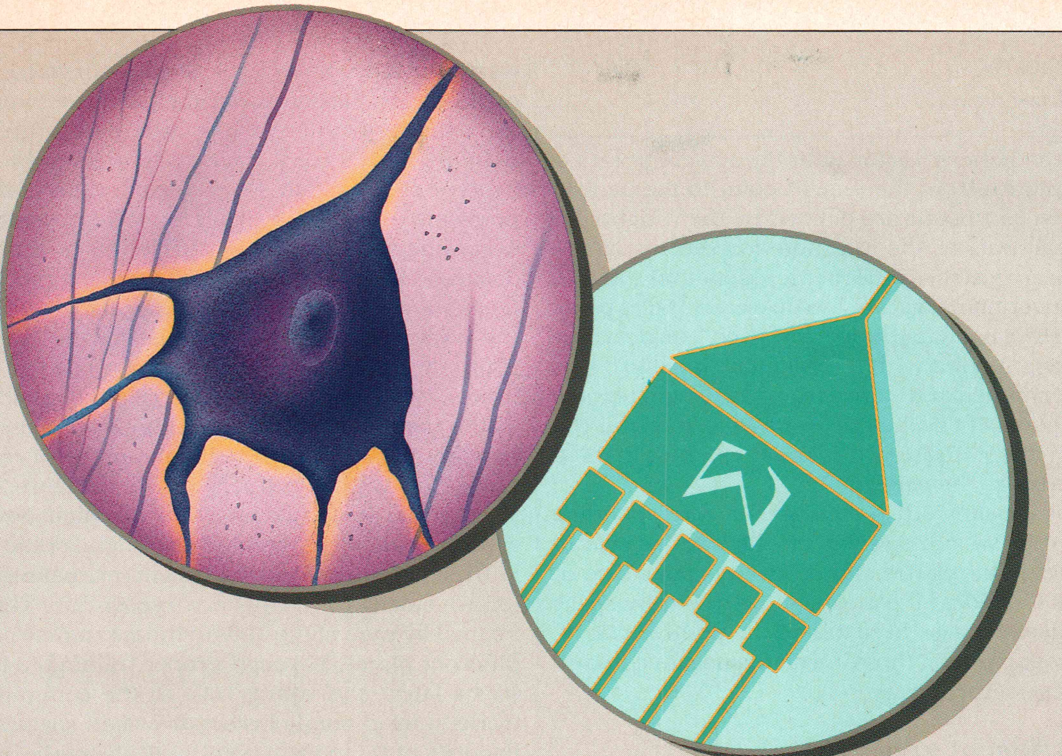
Robert Jay Brown, 5225 N.W. Ct., Margate, FL 33063. Robert is a graduate student at Florida Atlantic University. He is currently a consultant involved in designing electronic surveillance intercept and cryptography systems.

Neurophysiology

SILOAM (simple image learning on adaptive machinery) operates by modeling one possible organization of the actual neural structure of the brain on a computer. Consider a single nerve cell (see Figure 1, page 18). During World War II, a physiologist and a mathematician worked together to try to create a model of the brain based on anatomical and physiological experimental findings. The team of McCulloch and Pitts started by modeling the single nerve cell, or neuron. Their model is known today as the McCulloch-Pitts neuron and is depicted in Figure 2, page 18.

Microscopic studies reveal that the nerve cell is composed of a cell body, or cyton; many input fibers, or dendrites; and a single output fiber, or axon, which branches to send signals to the dendrites of other nerve cells. Physiological studies in which scientists selectively stimulate sets of dendrites while observing the axon yield the following result: some dendrites tend to cause the neuron to produce an output (the neuron "fires"), and other dendrites tend to prevent the neuron from firing. The dendrites that encourage firing are called "excitory" inputs, and those that discourage firing are called "inhibitory" inputs.

The cell either fires or does not fire. There is no specific tie between which inhibitory input cancels an excitory input. Each input has a "weighting factor" associated with it: excitory inputs have a positive weight, and inhibitory inputs have a negative weight. The neuron combines all these inputs (probably within the cyton) to produce the output. Thus it appears that the neuron adds up all the weights associated with inputs that are stimulated, and if the result exceeds a certain threshold, then the neuron fires.



Threshold Logic

This model of a neuron (Figure 3, page 18) is called a threshold logic unit (TLU). It is not difficult to construct a TLU out of readily available hardware components: a Schmidt trigger connected in series after an operational amplifier wired to operate as an analog summer will suffice (see Figure 4, page 19). The weights are the gains determined by the proportionality factors of the scaling resistors at the plus or minus inputs to the op-amp. Adjustable weights can be realized by using potentiometers with the wiper connected to the stimulus input, with one end of the resistance element connected to the plus input of the op-amp and the other end connected to the minus input. I chose to simulate the operation of a TLU with software. It's cheaper, and as you shall see, it allows the program to adjust its own weighting factors, which is necessary if the TLU is to be trained automatically rather than "tweaked" into alignment by a skilled technician.

The TLU is the physical embodiment of a linear equation formed by setting an inner product, or metric (measure of distance), to 0. The solution set for the equation thus formed defines a cleaving plane that acts as the boundary between two half-spaces in the weight space of all possible weight sets that could make up the weights for the input of a TLU. The cleaving planes, being the solution to a homogeneous linear equation, must pass through the origin. When the equality is changed to an inequality, the sign of the inequality indicates on which side of the cleaving plane the weight point lies.

SILOAM contains many TLUs. Each has as many inputs as there are pixels in the graphical array that is being examined by the network. In addition, each TLU has an additional input that is always excited. This extra input provides a "reference point", or "bias" (analogous to the DC component of a Fourier series), to set the threshold that determines the firing point of the TLU. This input is necessary to homogenize the linear equation formed by the inner product used to compute the metric. Without it, an input of all 0s would be degenerate and the training algorithm would fail to converge.

Pattern-Space Geometry

The task of computing the output of a TLU is performed by the vector operation of taking the "dot product" of the stimulus vector and the weight vector associated with the TLU. The sign of the result determines the output: a positive sign means the nerve has fired; a negative sign means it hasn't. Each weight vector can be interpreted as a point in hyperspace, or weight space. If you look at the dot product formed between the input pattern vector and the TLU weight vector, you see that, if the elements of the pattern vector are viewed as constants and the weight vector elements are viewed as variables, then if the dot product is set equal to 0, this dot product forms a linear homogeneous equation. The situation in 3-space is:

$$ax+by+cz = 0$$

A, b, and c are the components of the weight vector, and x, y, and z are the components of the pattern to be recognized. Remember that z is set to a constant value of 1. The solution set defines a plane that passes through the origin (see Figure 5, page 19). The plane forms a pattern surface that cleaves weight space into two half-spaces: this places one set of possible TLU weights on one side of the pattern plane and one set on the other side. Any given weight point will be on either the negative or the positive side of the pattern plane. (The two's complement arithmetic of the computer makes it convenient to consider a point lying on the plane itself to be on the positive side of the plane.)

The absolute value of the dot product is proportional to the perpendicular distance from the weight point to the pattern plane. Thus a given pattern hyperplane divides weight hyperspace into two half-hyperspaces. The dot

(continued from page 17)

product of the pattern vector with the weight point returns the distance from the weight point to the pattern plane. The weight points are defined by the weights for each of the inputs to the TLU; the coordinates of the weight point are just the values of the weights for the TLU.

By this convention, you can visualize each TLU as being represented by a point in this weight space. The dot product of the weight point with the augmented pattern vector is the perpendicular distance from the weight point to the pattern plane. If this quantity is positive, then the TLU "recognizes" the pattern; if it is negative, then the TLU does not recognize the pattern.

The TLU is a pattern dichotomizer: its corresponding weight point in weight space is on the positive side of some pattern hyperplanes and on the negative side of others. Thus it divides all pattern planes in weight space into two classes, or sets: those it is on the positive side of, which it recognizes, and those it is on the negative side of, which it does not recognize.

Artificial Neural Networks

Now, how about more difficult cases, in which a simple linear function cannot separate the categories? I will first explore a yes/no decision from a single network and will use one possible network of TLUs, called the committee network (see Figure 6, page 19). The committee network is a type of layered machine. A committee of TLUs is composed of an odd number of TLUs, each presented with the same input pattern vector. Each TLU in the committee decides whether the pattern is one that it recognizes, and it casts a vote accordingly. A chairman TLU then counts the votes. The chairman's inputs are the outputs of the committee members. Its weights are fixed at 1 for all its inputs, so it simply functions as a vote counter for the rest of the committee. By training multiple networks independently, you can increase the number of recognizable classes to any power of 2, as shown in Figure 7, page 20.

I have developed a democratic machine: how can such a thing work? I'll now show how a democratic committee with a majority-rule voting system can make a substantial improvement to my pattern recognizer. The divisions formed by each of the TLUs in the committee can occur at different places. Through proper training (which I'll describe later), the voting TLUs in the committee can be made to form point clusters in weight space such that a majority of TLU weight points will always be on the proper side of every presented pattern hyperplane. Thus you

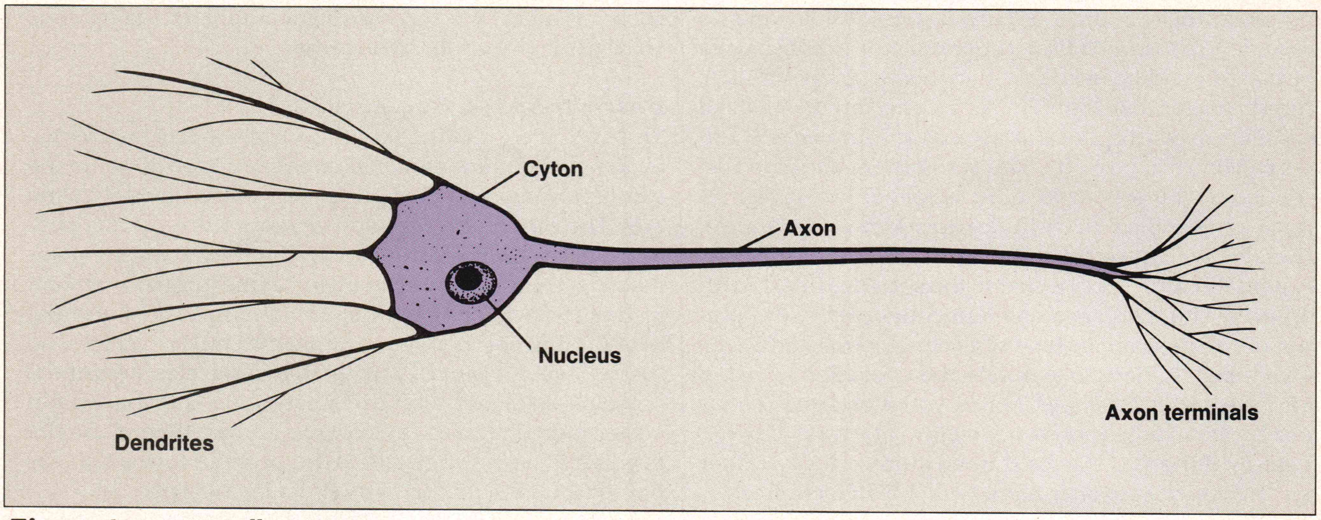


Figure 1: *A nerve cell, or neuron*

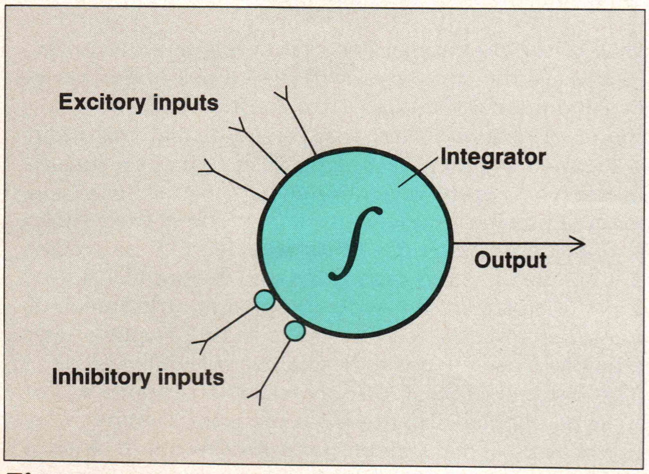


Figure 2: *McCulloch-Pitts neuron*

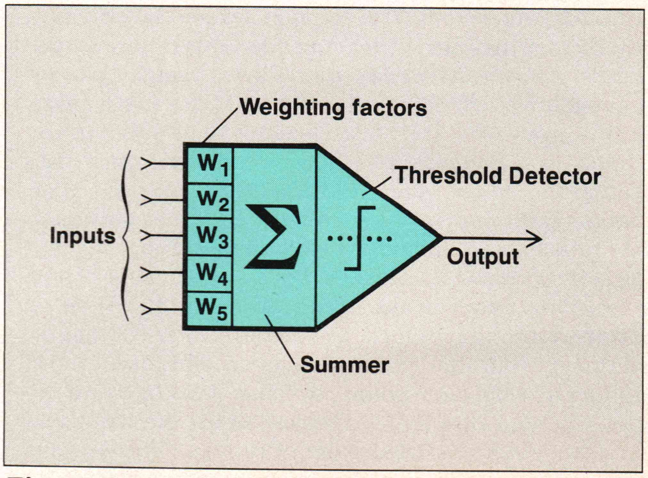


Figure 3: *The threshold logic unit*

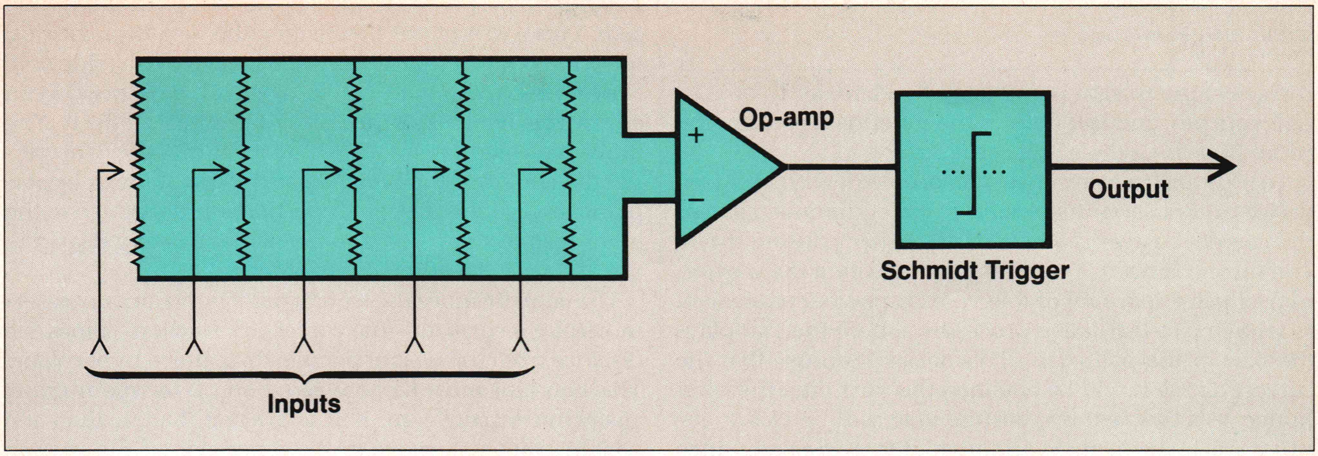


Figure 4: *Simplified hardware implementation of a threshold logic unit with adjustable weights*

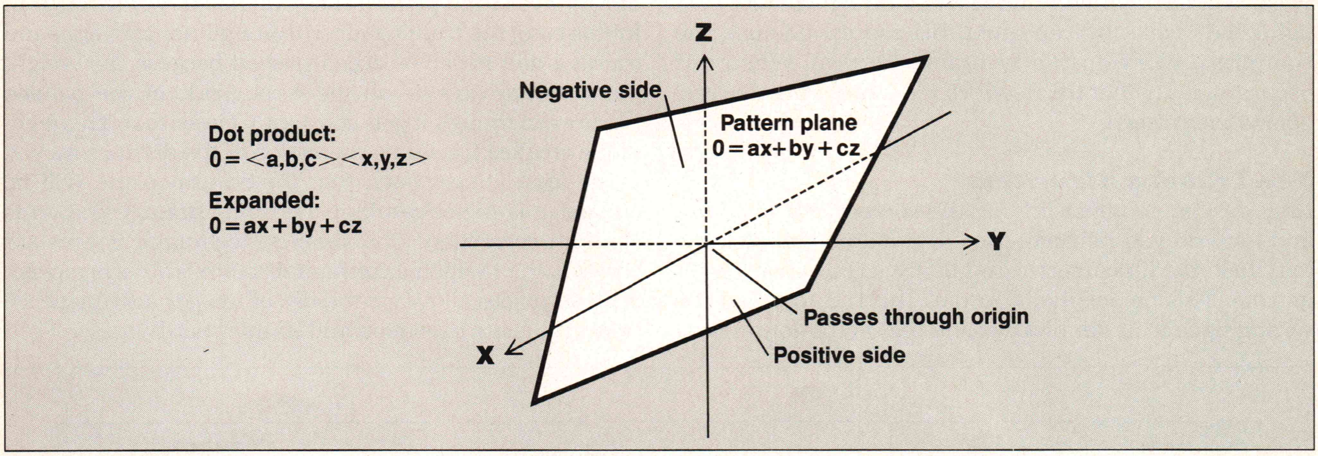


Figure 5: *The pattern plane as a linear homogenous equation derived from a dot product*

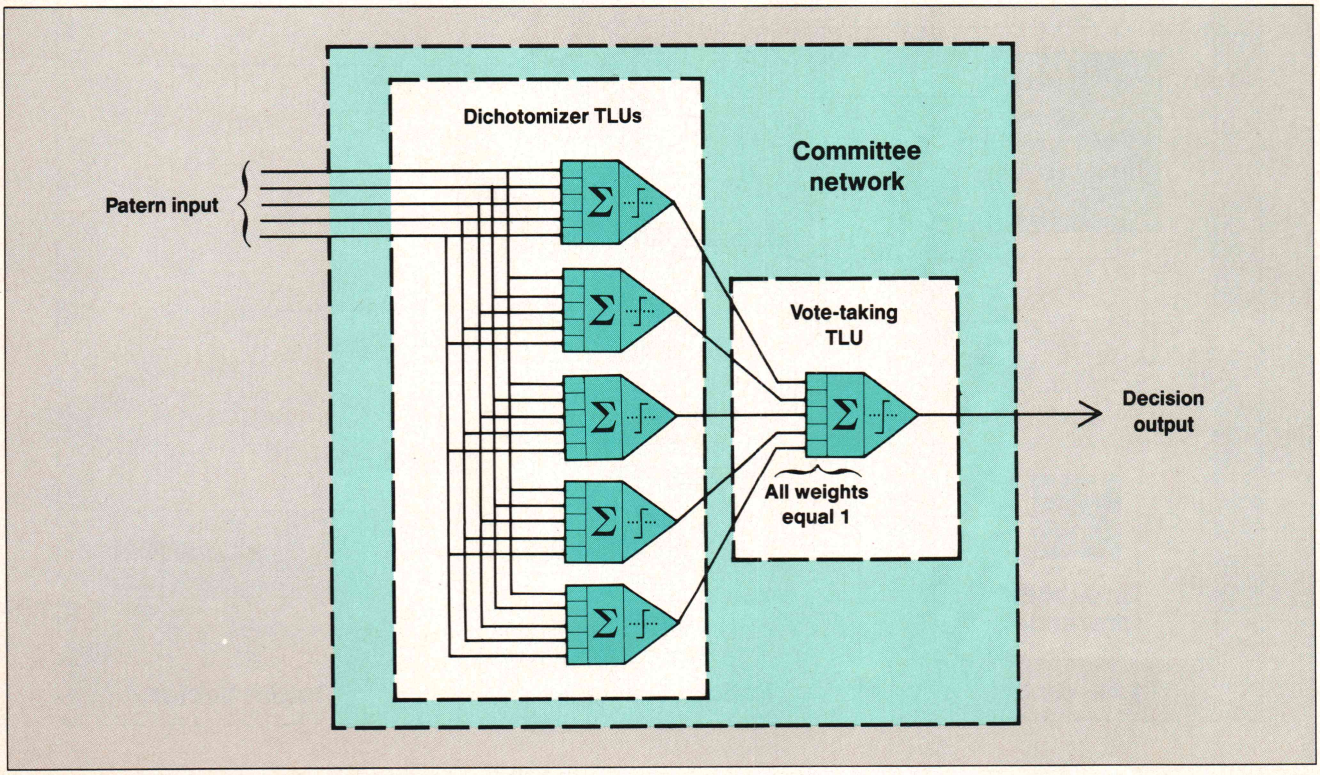


Figure 6: *A committee network of threshold logic units*

(continued from page 18)

can select the weight points such that a majority of TLUs will vote yes, and this defines the set of patterns that the committee as a whole will recognize.

To train such a committee, you present it with a pattern vector and observe the result. If the committee returns the correct answer, you present another pattern; if not, you must correct it. You do this by adjusting the weights to produce a more favorable vote (somewhat equivalent to lobbying in legislative processes, where the TLU plays the part of the politician). This does not ensure that the correct decision will be obtained the next time the committee sees this pattern, but the vote will be closer. Because you insist on an odd number of TLUs in a committee (exclusive of the chairman), you can never have a tie. What you do is convert one TLU at a time to a more "enlightened" view. By repeating this process enough, the committee will return a favorable decision. When this occurs, you say that the network has been trained to recognize the pattern.

The Training Algorithm

How do you go about finding the correct TLU to adjust, and how do you perform the adjustment? You pick the TLU that voted incorrectly and that was the least sure of its vote. This means that you pick the TLU that had the wrong sign for its dot product and that had a dot-product magnitude that was the least of all TLUs with the wrong sign. This corresponds to selecting the weight point closest to the pattern hyperplane and on the wrong side of it. Now you know which TLU to work on, but how do you adjust the weights to produce the desired effect? You move the weight point for the selected TLU along the perpendicular from the weight point to the pattern hyperplane toward the pattern hyperplane and through to the other side of the pattern hyperplane, thereby changing the TLU's classification of the pattern.

You actually move the weight point by an amount determined by a constant—the correction fraction—times the distance from the weight point to the pattern hyperplane. This constant must lie between 1 and 2 for the training algorithm to converge. If it is greater than 1, then the weight point will move to the other side of the pattern hyperplane; if it is less than 1, then the weight point will move toward the pattern hyperplane but not through it. In this case, the training algorithm will not converge and training will never be accomplished because the weight point will always be on the wrong side of the pattern plane even though it gets constantly closer to it. This technique is called fractional correction. If the distance moved is the least integer such that the pattern plane will be crossed, this choice results in a training strategy known as absolute correction. The simplest technique is constant correction, in which a constant distance is always moved. Such strategies allow for the use of integer arithmetic resulting in faster execution and simpler hardware.

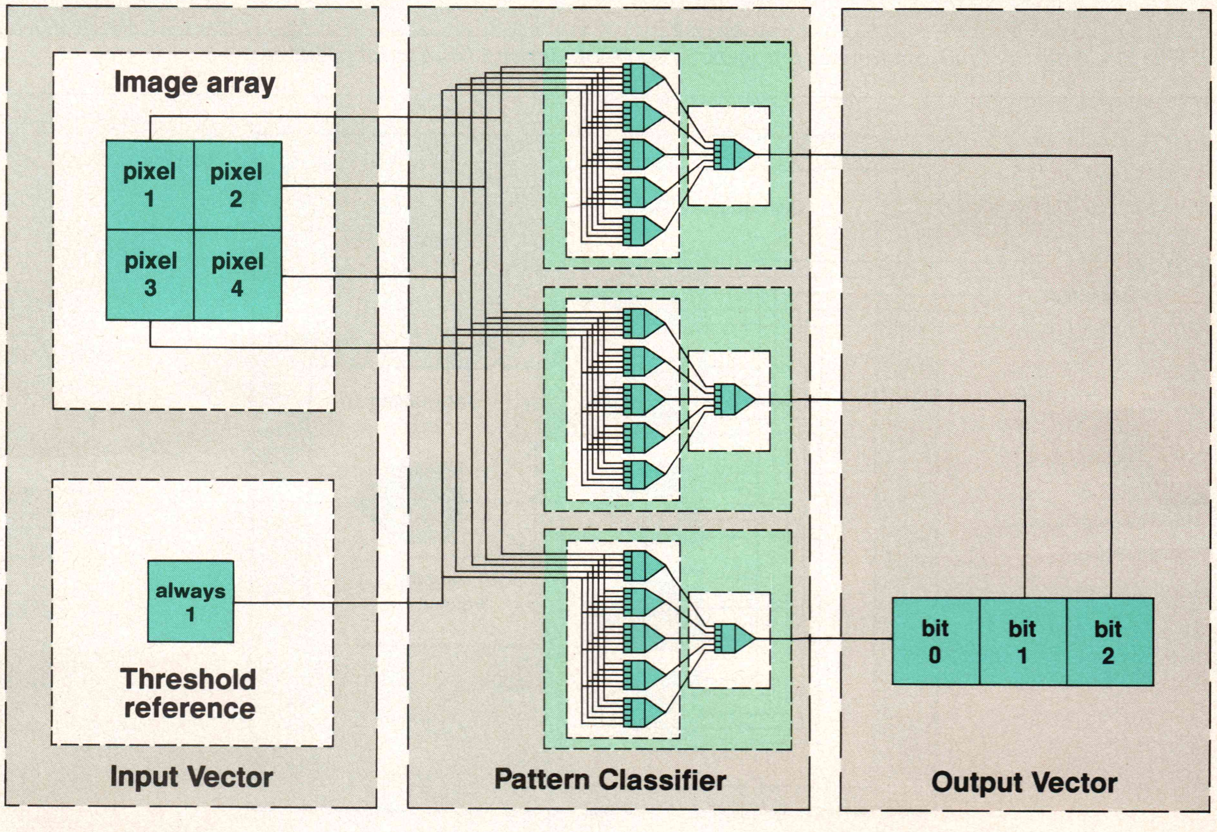


Figure 7: *An image learning system to classify 2^n patterns*

New Turbo Prolog and Turbo Pascal Toolboxes add more Power

New! Turbo Prolog Toolbox

Our new Turbo Prolog Toolbox™ enhances Turbo Prolog—with more than 80 tools and over 8,000 lines of source code that can easily be incorporated into your programs. It includes about 40 example programs that show you how to use your new tools.

New Turbo Prolog Toolbox features include:

- Business graphic generation
- Complete communications package
- File transfers from Reflex, dBASE III, 1-2-3, Symphony
- A unique parser generator
- Sophisticated user-interface design tools

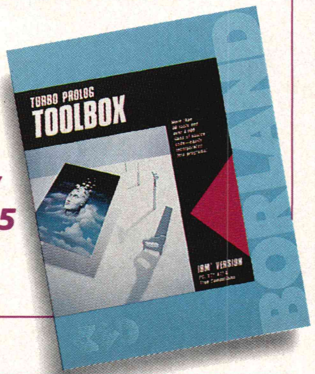
System requirements

Turbo Prolog: IBM PC, XT, AT or true compatibles. PC-DOS (MS-DOS) 2.0 or later. 384K. Turbo Prolog Toolbox requires Turbo Prolog 1.10 or higher. Dual-floppy disk drive or hard disk. 512K.

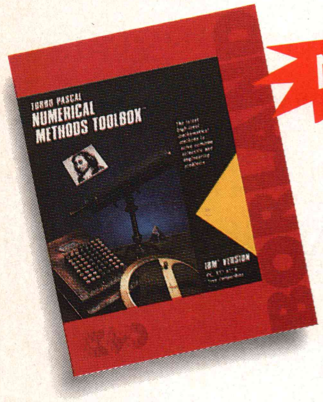
It's the complete developer's toolbox and a major addition to Turbo Prolog. You get a wide variety of menus—pull-down, pop-up, line, tree and box—so you can choose the one that suits your application best. You'll quickly and easily learn how to produce graphics; set up communications with remote devices; read information from Reflex,® dBASE III,® Lotus 1-2-3® and Symphony® files; generate parsers and design user interfaces. All of this for only $99.95.

NEW

Only $99.95



The power and high performance of Turbo Pascal® is already in the hands of more than half-a-million people. The technically superior Turbo Pascal is the *de facto* worldwide standard and the clear leader.



NEW

Only $99.95

New! Turbo Pascal Numerical Methods Toolbox

What our new Numerical Methods Toolbox™ will do for you now:

- Find solutions to equations
- Interpolations
- Calculus: numerical derivatives and integrals
- Fourier transforms
- Matrix operations: inversions, determinants and eigenvalues
- Least squares approximations
- Differential equations

As well as a free demo FFT program, you also get Least Squares Fit in 5 different forms.

1. Power
2. Exponential
3. Logarithm
4. 5-term Fourier
5. 5-term Polynomial

They're all ready to compile and run.

All this for only $99.95

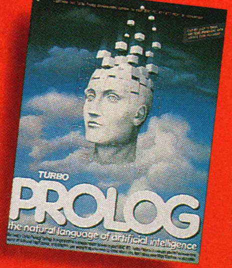
System requirements

IBM PC, XT, AT or true compatibles. PC-DOS (MS-DOS) 2.0 or later. Turbo Pascal 2.0 or later. Graphics module requires graphics monitor with IBM CGA, IBM EGA, or Hercules compatible adapter card, and requires Turbo Graphix Toolbox. 8087 or 80287 numeric co-processor not required, but recommended for optimal performance. 256K.

Turbo Pascal 3.0: Includes 8087 & BCD features for 16-bit MS-DOS and CP/M-86 systems. CP/M-80 version minimum memory: 48K; 8087 and BCD features not available. 128K.

Turbo Prolog, the natural language of Artificial Intelligence

Turbo Prolog™ is the most popular AI package in the world with more than 100,000 users. It's the 5th-generation computer programming language that brings supercomputer power to your IBM® PC and compatibles. You can join the AI revolution with Turbo Prolog for only $99.95. Step-by-step tutorials, demo programs and source code included.



Only $99.95!

"If you're at all interested in artificial intelligence, databases, expert systems, or new ways of thinking about programming, by all means plunk down your $100 and buy a copy of Turbo Prolog."
Bruce Webster, BYTE

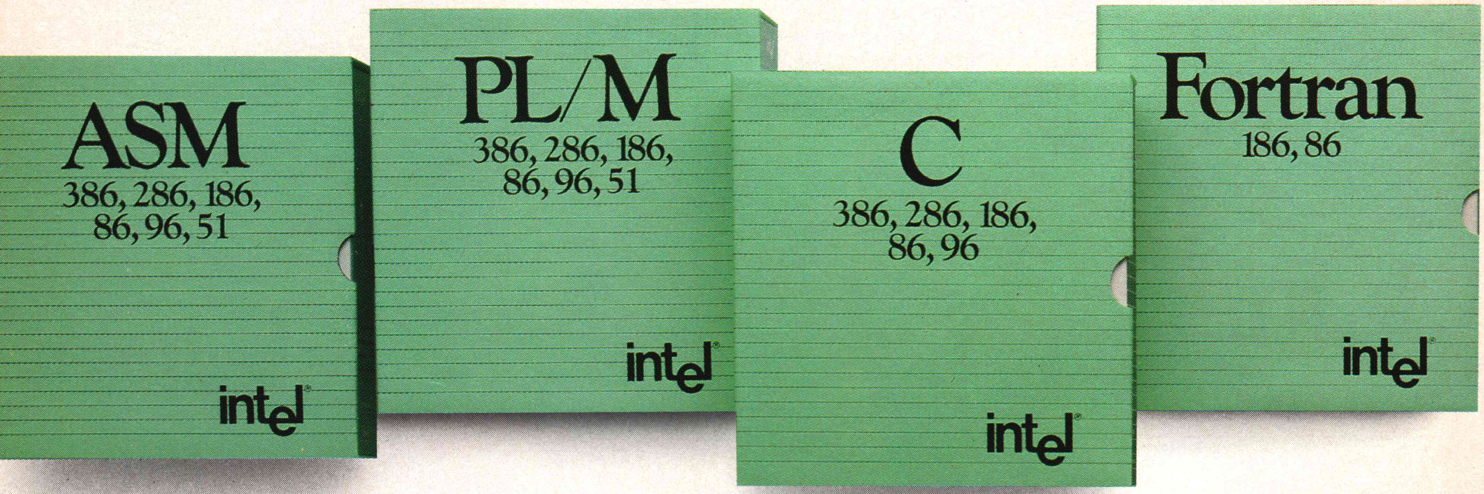
For the dealer nearest you, or to order by phone
call (800) 255-8008
CA (800) 742-1133
Canada (800) 237-1136

BORLAND
INTERNATIONAL

4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

Circle **no. 241** on reader service card.

WRITE FASTER IN ANY LANGUAGE.



If you develop software for any product based on an Intel microcontroller or microprocessor, including the 80386, the unique debug hooks in the Intel languages will help get the job done faster.

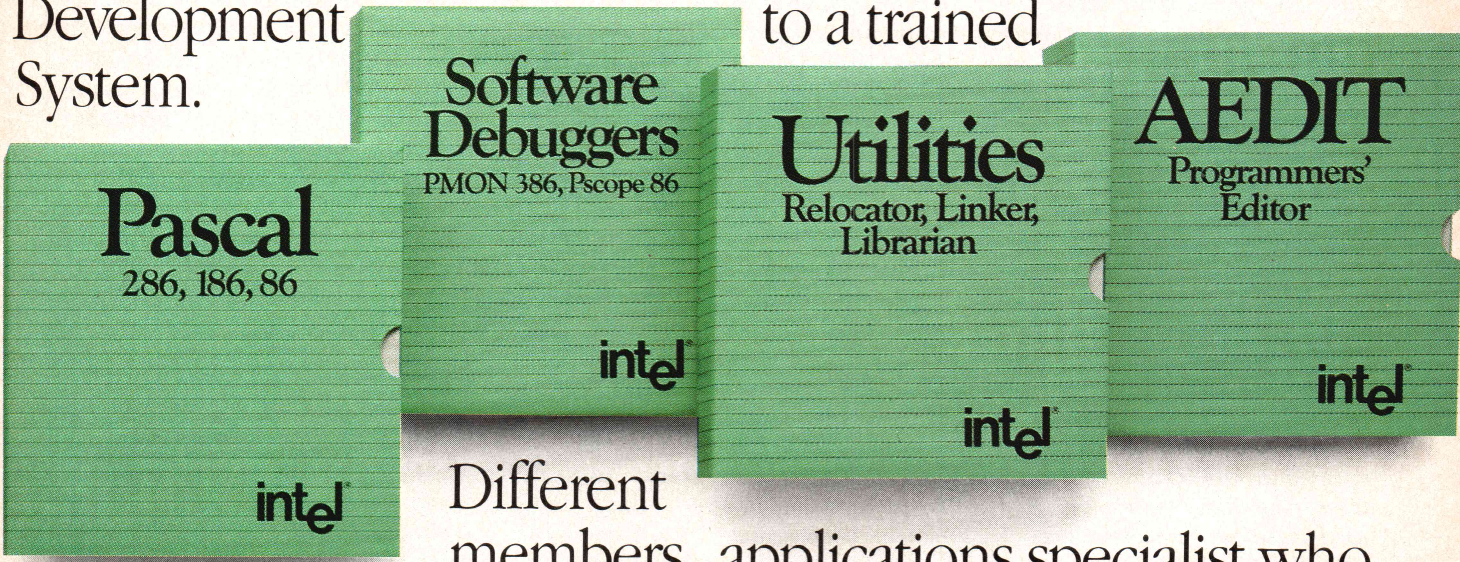
In fact, when used with Intel debuggers and emulators, Intel development languages can provide more debug data than any other high-level language.

Debug hooks let you symbolically debug in the same high-level language you wrote in without having to deal with machine or hex code. Which means 80.386 reads as 80.386, not 50 62 D0 C5.

Because the location of both code and data are easily specified with our locator, it is easier for you to develop ROM-based firmware.

Since Intel languages

produce identical object code regardless of the host, you can write code at a PC running DOS, a VAX*/VMS terminal, or an Intel Development System.



Different members of the same design team can therefore choose the most effective combination of languages and systems to get the job done faster.

Intel post-sales support can also help you get the job done faster. We invented the microprocessor. We know microprocessors and languages for Intel architectures better than anyone else.

When you buy an Intel language, you have access to our customer hotline. So if you ever have a question you can talk directly to a trained applications specialist who understands our products. And can give you the right answers. Faster.

To order today, or get more information—including a free catalog of our development tools—call toll-free 1-800-87-INTEL.

The sooner you call, the faster you'll get the job done.

intel®

(continued from page 20)

Fixed-increment correction with binary images using 8-bit signed integer weights lends itself to cheap parallelism. A separate Intel 80C51 microcomputer on a chip could be used for each TLU, taking weight points out of an array in on-board ROM. Using six committees of seven voting TLUs each, plus a single 80C51 to count all the votes and act as central control, results in $6 \times 7 + 1 = 43$ 80C51 processor chips. A system such as this should be able to perform real-time optical text scanning of printed literature. Using surface-mount or hybrid packaging methods, the device might be as portable as a Walkman. A speech synthesizer could serve as an output device, receiving ASCII text from the pattern recognizer. Voilà!—a reading device for the visually handicapped. To this end, I have examined the performance of an 8-bit integer network with fixed-increment correction with encouraging results. The TMS-320C25 digital signal processor from Texas Instruments may prove less expensive than the 80C51 array. Although the 320 is more expensive, it is much faster than the 80C51, and the overall system cost may be less.

The Experiment

There are actually three versions of SILOAM—a floating-point version, a 16-bit integer version, and an 8-bit integer version (see Listing One, page 56). The symbol *ELTYPE* is defined on the compiler invocation line and determines the type definition for an element of a weight point vector. You select the pattern presentation order for training with the *−o* option and select initial conditions for the weight points with the *−r* option. The correction method is selected with *−a*, *−i*, and *−f* options, which specify absolute, fixed increment, or fractional correction, respectively. You select the level of detail for logging with the *−l* option: *−l0* displays only final results; *−l3* displays the most detail.

I ran the program on a small pattern file, representing a binary image of each of the numerals 0−9 (Table 1, below). It has also successfully learned the entire uppercase alphabet. The alphabet pattern file was generated by rasterizing characters from the Hershey character database of the National Bureau of Standards. The entire ASCII character set was generated in this fashion as a high-resolution dot-matrix raster and was taught to SILOAM. The output produced by a run of the program with the high-resolution sample character set is shown in Example 1, below.

It is interesting that a network of only one TLU per committee could learn all of the binary images of the character sets. When this is the case, the pattern set is said to be "linearly separable." A pattern file of random analog pixel values was generated, comprising 100 images. In this case, a single TLU could not learn the pattern set—three TLUs were required, and different training methods produced radically different results. Fixed increment performed quite poorly, absolute correction did somewhat better, but fractional correction did the best. Values of the correction fraction closer to 2 seemed to perform better and resulted in faster convergence. In fact, convergence was even achieved with values greater than 2, although when they got up to about 2.5, convergence failed.

Limitations

SILOAM can actually get confused and forget things it has already learned in the process of trying to learn new things. In *Learning Machines*, Nilsson shows, however, that the training procedure will converge to the desired result, given that you choose a suitable distance to move the weight point and that you do not exceed the capacity of the machine (related to the number of TLUs per committee and the number of patterns to be recognized). This result is known as the Fundamental Training Theorem, or the Perceptron Convergence Theorem.

Despite the impressive performance of this simple network, it has some serious theoretical shortcomings—for example, it cannot learn a simple exclusive-OR function.

```
0. 1. 0. 0.   0. 1. 1. 0.   1. 1. 1. 0.   1. 0. 0. 0.
1. 1. 0. 0.   1. 0. 0. 1.   0. 0. 0. 1.   1. 0. 1. 0.
0. 1. 0. 0.   0. 0. 1. 0.   0. 1. 1. 0.   1. 1. 1. 1.
0. 1. 0. 0.   0. 1. 0. 0.   0. 0. 0. 1.   0. 0. 1. 0.
1. 1. 1. 0.   1. 1. 1. 1.   1. 1. 1. 0.   0. 0. 1. 0.
1             2             3             4

1. 1. 1. 1.   0. 1. 1. 1.   1. 1. 1. 1.   0. 1. 1. 0.
1. 0. 0. 0.   1. 0. 0. 0.   1. 0. 0. 1.   1. 0. 0. 1.
1. 1. 1. 0.   1. 1. 1. 0.   0. 0. 1. 0.   0. 1. 1. 0.
0. 0. 0. 1.   1. 0. 0. 1.   0. 1. 0. 0.   1. 0. 0. 1.
1. 1. 1. 0.   0. 1. 1. 0.   0. 1. 0. 0.   0. 1. 1. 0.
5             6             7             8

0. 1. 1. 0.   0. 1. 1. 0.
1. 0. 0. 1.   1. 0. 0. 1.
0. 1. 1. 1.   1. 0. 0. 1.
0. 0. 0. 1.   1. 0. 0. 1.
1. 1. 1. 0.   0. 1. 1. 0.
9             10
```

Table 1: *Binary images of numerals*

```
Invoked By: ascii -i1 -t1
element type is int

initializing

mean of the radii: 31.749012
standard deviation: 0.001284

training completed in 5771 seconds.

number of committees: 7
number of tlus total: 7
number of elements: 7063
number of connections: 6302

number of passes thru file: 78
number of patterns in file: 87
number of mis-recognitions: 1190
number of tlu adjustments: 1190
maximum element magnitude: 49.000000

mean of the radii: 83.3047640
standard deviation: 0.582814
```

Example 1: *Output of SILOAM with high-resolution character set*

Clarify your source code

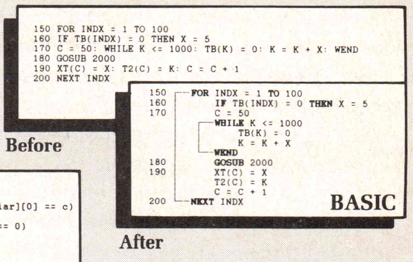
C, BASIC, Pascal, dBASE, Modula-2 programmers: be more productive with two new utilities from Aldebaran Laboratories

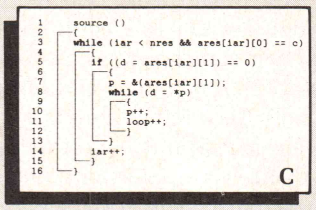
"Occasionally, a utility comes along that makes a programmer's life much easier. SOURCE PRINT is such a program. It contributes to the programmer's job by organizing code into a legible format and by helping to organize the documentation and debugging process."

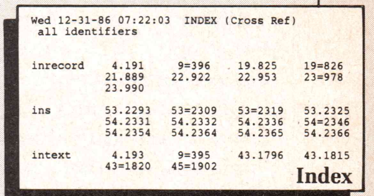
— PC Magazine
Sept. 16, 1986

Source Print™

organizes your source code, simplifies debugging, and makes documentation a snap! It lists one or more source files with informative page headings and optional line numbers, while offering invaluable features:

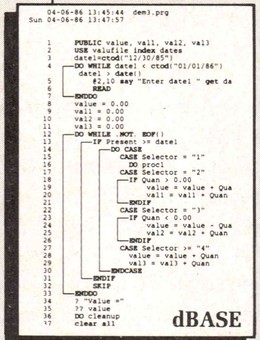






The Index (Cross-Reference list) saves you time by showing exactly where variables are used and where functions, procedures, and routines are called. Locations where new values may be assigned to variables are shown, making it easy to track down that mysterious value change.

$75.00



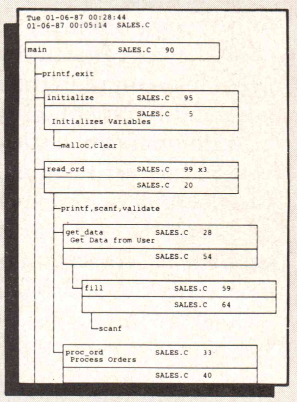
Structure Outlining solves the problem of hard-to-see nested control structures by automatically drawing lines around them.

Automatic Indentation of source code and listings reduces your editing time and ensures indentation accuracy.

Plus... Source Print generates a table of contents listing functions and procedures. Keywords can be printed in boldface on most printers. Multi-statement BASIC lines can be split for readability. Functions and procedures can be drawn by name from one or more source files to form a new file.

Tree Diagrammer™

shows your program's overall organization at a glance. Ordinary program listings merely display functions, procedures, and subroutines sequentially, but do not display the relationships between these routines. Our revolutionary new Tree Diagrammer automatically creates an "organization chart" of your program showing the hierarchy of calls to functions, procedures, and subroutines. Recursive calls are indicated and designated comments in the source code will appear on the chart.



Tree Diagrammer helps you organize your program more logically. And you'll be amazed at how easy it is to debug when you see how your routines interact.

$55.00

Source Print and Tree Diagrammer both have easy-to-use menus with point-and-shoot file selection, and let you search for files containing a given string. For IBM PC and compatibles with 256K.

Join thousands of programmers who are working more efficiently using Source Print and Tree Diagrammer. Order these indispensable tools today. We ship immediately, and there's no risk with our 60-day money-back guarantee. **Order both and save. Only $115.00.**

800-257-5773 Dept. 54
In California:
800-257-5774 Dept. 54

MasterCard, VISA, American Express, COD. Add $5 for shipping.

Source Print and Tree Diagrammer are trademarks of Aldebaran Labs. dBASE is a trademark of Ashton Tate.

Aldebaran Laboratories 3339 Vincent Rd. Pleasant Hill, CA 94523 415-930-8966

YES! Rush me ☐ Source Print @ $75. ______ ☐ Tree Diagrammer @ $55. ______
☐ Both $115. Ship/Handling $5. For CA add 6% tax ______ Total ______
Name ______
Company ______
Address ______
City ______ State ______ Zip ______
☐ Check enclosed ☐ VISA ☐ MasterCard ☐ American Express
Card # ______ Exp. Date ______
Signature ______ Phone # ______ 54

Circle **no. 350** on reader service card.

(continued from page 24)

This deficiency can be shown with a geometric proof. Because there are two variables input to an XOR, a 2 × 1 input pattern is needed. This results in three dimensions for the pattern space. Because there are four combinations of the inputs, there are four pattern planes. All four planes pass through the origin. If you imagine a sphere about the origin, these four planes intersect the sphere in four great circles and these four circles each intersect each other. If you place the south pole of the sphere on the origin of a 2-D graph and project the surface of the sphere onto the plane (as in a polar projection of the globe), four intersecting circles result—one centered in each quadrant of the plane. Their common intersection is around the origin of the graph. If you count the number of distinct regions into which these circles divide the plane, you get 14, but there are 16 possible combinations of accepting and rejecting 4 distinct patterns composed of 2 independent variables. The exclusive-OR and the exclusive-NOR, or equivalence relation, are these missing regions. You can use a similar argument in higher dimensions to show that all such networks are likewise deficient in not being able to learn all possible pattern sets with which they could be confronted.

In *Parallel Distributed Processing*, Rumelhart et al. show, however, that other network topologies, neural activation functions, and training algorithms—especially the gradient-descent training method—are capable of producing all possible Boolean switching functions. I highly recommend their book for anyone interested in pursuing the study of artificial neural networks in depth.

Topics for Further Investigation

One idea that needs to be explored is recognizing patterns over time—that is, sequences of patterns. One idea might be to incorporate some sort of feedback into the network to provide a memory capability. The output bit vector could be concatenated with the pattern input vector to provide an input to the network that was a function not only of the current input but also of the previous output.

Additional exploration could be done with nonbinary pattern elements. Remember that the recognition process does not require the elements of the pattern vector to be 1s and 0s; any real values will still satisfy the vector geometrical constraints and should be recognizable with essentially the same algorithm.

Geoffrey Hinton discussed some of his work with artificial neural networks at the AAAI-86 conference in Philadelphia last August (see bibliography). He showed how a real-valued, differentiable activation function could be trained by the method of gradient backflow to form its own independently developed internal abstractions. His experiment involved learning two similar family trees. The network formed, on its own and without being taught explicitly by the examples, the abstract concepts of generation and various family relations. It was also able to generalize its experience to determine the relations between individuals it had not been previously introduced to. It got better than three out of four new problems correct. Hinton showed that a statistical correlating recognizer would fail this test and that true generalization of induced abstractions was being performed.

Neurophysiology Revisited

What can you learn about the natural mind based on a model neural network? Psychiatrists treat mental illness with drugs, such as phenathiazines and lithium, and with electroshock therapy. Electroshock therapy is assumed to destroy connections, thereby altering weights by setting them to zero. Phenathiazines affect the release of neurotransmitters such as serotonin, norepinephrine, and dopamine. A change in these would have the same effect as altering the weights at the inputs to the neuron. Lithium is metabolized by the body's electrochemistry in the same manner as is sodium, but it behaves differently in nerve conduction and firing potential. Thus lithium acts as an inert placeholder for the neuroactive sodium in the sodium-potassium complex. The presence of lithium would affect the threshold of the neuron, but this is just another weight in my model. Thus these psychoactive drugs are trying to counteract a medical problem that is manifested in a perturbation of the weights of the neuron. If the drugs are underprescribed, the desired effect will not be achieved, and if they are overprescribed, the weights may be totally scrambled, resulting in a worsening of symptoms.

Hardware Implementations

Recently, threshold logic has been receiving a lot of attention from the press. The front page of the *Electronic Engineering Times* carried an article on February 3, 1986, entitled "Neural Research Yields Computer That Can Learn." This article described research into a speech learning program based on Hopfield networks that apparently makes use of time feedback techniques similar to those outlined here.

The next week an article appeared in the same publication that touted threshold logic as the key to optical computers. This article gave some details on how electro-optical technology can implement threshold logic gates with an enormous number of inputs.

The week after that, an article appeared giving details of a gallium arsenide TLU that uses the analog addition of the brightness of light waves to perform the summing operation, so that the device is essentially a threshold detector with a photoresistor for an input. This TLU implementation does not increase in complexity no matter how many inputs it recieves: they are just lights shining on its single photoresistor. The output of this device is a single solid-state laser. This last article also described a holographic optical interconnect scheme that is very interesting. The output lasers reflect off a hologram placed over the chip. The hologram acts as a phased array reflector that directs each output laser's light only to those photoresistors that are supposed to be connected to that output. In this way, the logic signals travel at the speed of light without wasting any chip real estate on signal interconnect lines. The logic can be placed closer together, and a 3-D medium is available for interconnect wiring instead of the 2-D masks of current-day chips. Gallium arsenide chips are now available that operate at speeds of 20 GHz. With more closely spaced circuits and more flexible de-

sign rules for interconnects, a tremendous increase in speed should be gained from these new devices.

This kind of hardware is just what is needed to take these learning systems from the several seconds per iteration speed zone into the picoseconds per iteration arena. You can certainly expect to see more threshold logic learning systems in the future if this hardware implementation effort succeeds.

Availability

All the source code for articles in this issue (except for C Chest) is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

Bibliography

Brown, Chappell. "Neural Research Yields Computer That Can Learn." *Electronic Engineering Times* (February 3, 1986).

Brown, Chappell. "Threshold Logic Gates: Key to Making Optical Computers Reality?" *Electronic Engineering Times* (February 10, 1986).

Brown, Chappell. "Researchers See Lightwaves at End of VLSI Tunnel." *Electronic Engineering Times* (February 17, 1986).

Buzan, Tony, and Dixon, Terence. *The Evolving Brain*. New York: Holt, Rinehart & Winston, 1977.

Hinton, Geoffrey E. "Learning Distributed Representations of Concepts." *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Amherst, Mass.: Lawrence Erlbaum Assoc., August 1986.

Hopfield, J. J. "Neural Networks and Physical Systems with Emergent Computational Abilities." *Proceedings of the National Academy of Sciences* 79 Washington, D.C., 1982.

Masland, Richard H. "The Functional Architecture of the Retina." *Scientific American*, vol. 255, no. 6 (1986).

McCulloch, W., and Pitts, W. "A Logical Calculus of the Ideas Immanent in Nervous Activity." *Bulletin of Mathematical Biophysics* 5 (1943).

Miller, Richard K. *Optical Computers: The Next Frontier in Computing*. Madison, Ga.: SEAI Technical Publications, 1986.

Nilsson, Nils J. *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. New York: McGraw-Hill, 1965.

Rumelhart, David E.; McCleland, James L.; and the PDP Research Group. *Parallel Distributed Processing*. Cambridge, Mass.: MIT Press, 1986.

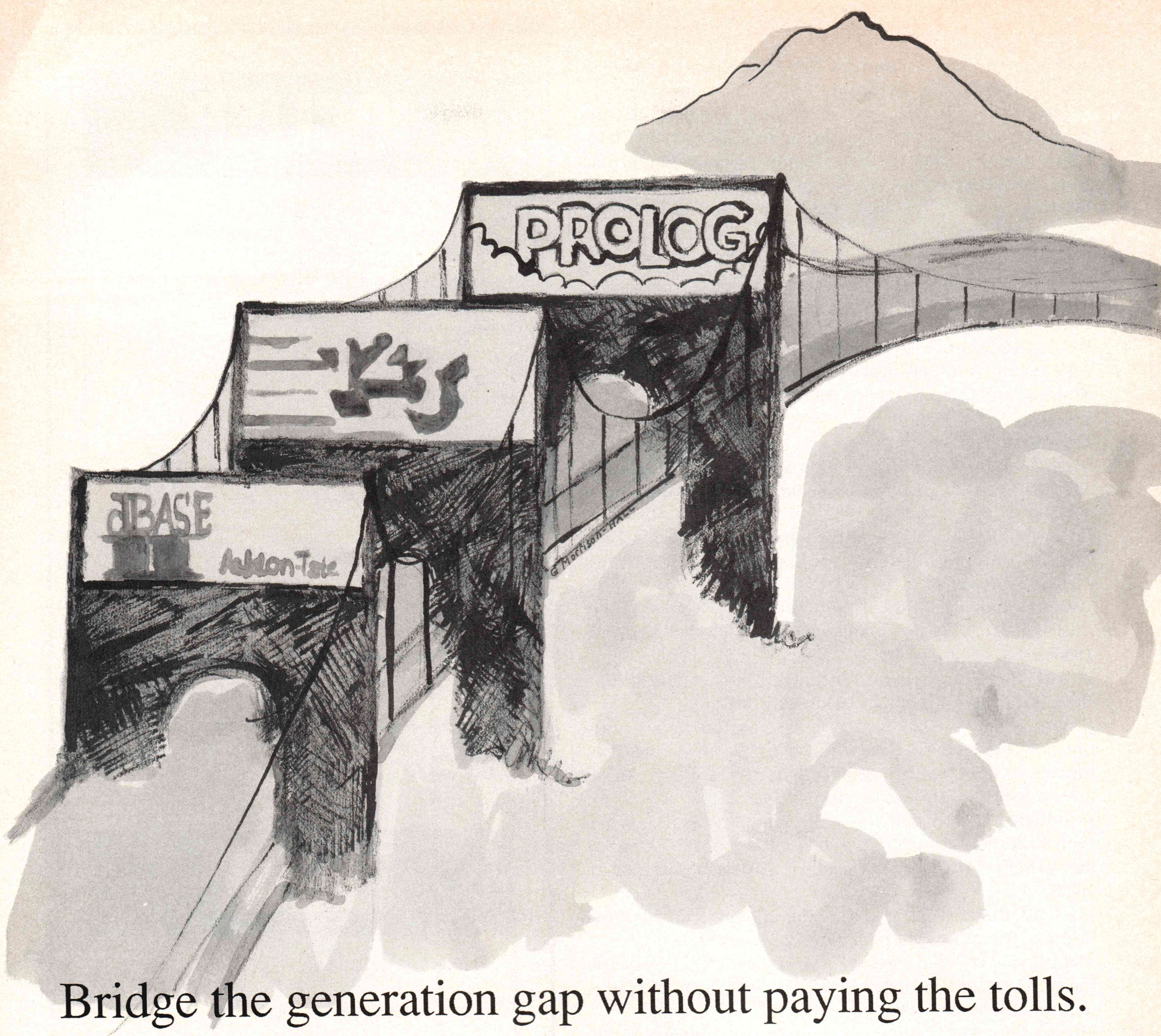
Schwartz, Tom J. "IBM Research Yields Artificial Neural Net Workstation." *Electronic Engineering Times* (September 1, 1986).

Silbar, Margaret L. "In Quest of a Human-like Robot." *Analog* 88 (November 1971).

DDJ

(Listing begins on page 56.)

Vote for your favorite feature/article.
Circle Reader Service **No. 2.**



Bridge the generation gap without paying the tolls.

BridgeWare™ links the fifth generation power of Turbo Prolog with fourth generation databases, spreadsheets and languages. Now you can create AI applications, expert systems and decision support systems that work with the programs and information you already have.

• uses pop-up windows and menus to graphically model complex data structures and file combinations

• easily combines data derived from several files and produced by different programs or languages into one or more Prolog structures

• compatible with dbase II and III, Lotus 123 and Symphony, C, Pascal, BASIC, and virtually any high level language for the PC

• includes examples combining dbase, Lotus and Turbo Prolog; complete with source and databases

• comprehensive and easy to use documentation

Use the power of BridgeWare™ to turn databases into knowledge bases. Price: $69.95 includes handling and shipping within the US. PA add 6% sales tax. Order toll free, 24 hours at 1-800-351-5858 ext 409. In PA 215-934-3967.

MicroBase™
Software Systems

MicroBase Software Systems, Inc.
3238 Red Lion Road
Philadelphia, PA 19114

The following are trademarks: BridgeWare and MicroBase—MicroBase Software Systems, Inc.; Turbo Prolog—Borland International; dbaseII and dbaseIII—Ashton Tate; Lotus 123 and Symphony—Lotus Development Corp.

Circle **no. 150** on reader service card.

ACTOR WILL DO WINDOWS.

If you've been wondering how you're going to explore Microsoft® Windows and then work it into your programs, what you need has just arrived.

ACTOR™

ACTOR is a new programming system, and there are two reasons why it's so good with Windows.

First, ACTOR is an ideal programming environment. It shows you all your work through Windows. Any part of the program you're writing, its output as it runs, error diagnostics and tracing, catalogs of routines, and, in fact, just about anything you like—they can all be visible on the screen and accessible at the same time, in their own windows.

So it's easier to see what you're doing.

Second, ACTOR is an object-oriented programming language. Everything in an ACTOR program is a self-contained package with its own data and procedures. That is, an object. And the objects pass messages to each other while the program runs.

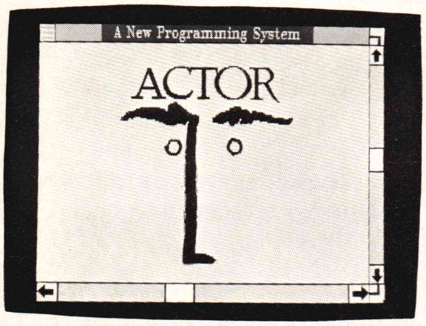
With ACTOR, you don't write lists of instructions. Instead, you define what classes of objects do. Which is especially easy because they inherit most of their behavior from other classes. That means there's less for you to worry about.

And when you create an object, you just choose its name and starting values. After that it pretty much takes care of itself.

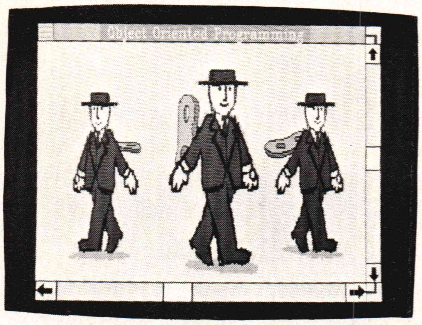
What's important here is that windows are objects like everything else in the language. You make as many as you need, and they do what they're supposed to, mostly on their own.

So it's natural and easy to put windows in your application programs. And users will benefit from Windows when your programs run, as much as you do when you write them.

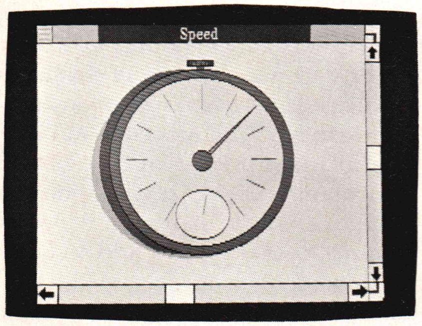
To order, just send in the coupon. Or call **312-491-2370** for more information. You'll see that ACTOR will do Windows, and more.



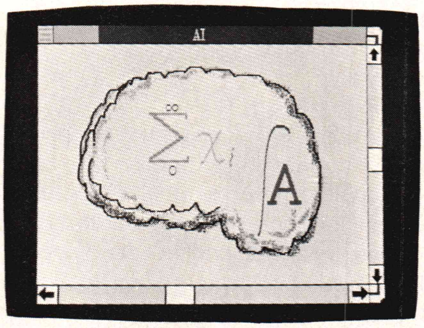
ACTOR™ is a new, interactive programming system, the first with Microsoft® Windows. You have Windows when you write a program, and users have Windows when they run it.



Moreover, ACTOR is a new, object-oriented programming language. Winding objects up and turning them loose is a lot easier, more productive, and more fun than old fashioned programming.



Thanks to a new method of "incremental garbage collection," ACTOR never has to slow down to clean up memory like other advanced languages. You can even use it for real-time control. It's that fast.



ACTOR offers all the features of an ideal artificial intelligence language, but in a familiar, Pascal-style syntax. Which makes artificial intelligence programming easier. And what could be smarter than that?

Please send me more information about ACTOR.

Name__________

Address__________

City__________ State_____ Zip_____

Daytime phone__________

☐ Check or money order enclosed. ☐ Bill my VISA. ☐ Bill my MasterCard.

Card number__________ Exp. date_____

☐ I'd like to place an order while I'm at it. (ACTOR is not copy protected and has a 60-day money back guarantee. Educational pricing is available. ACTOR requires an IBM PC/XT or AT or equivalent. 640K and hard disk is recommended.)

☐ __ copies of ACTOR with Windows runtime support at $495 each. $_____

☐ __ copies of ACTOR, including Windows complete with MS-DOS Executive, at $550 each. $_____

■ Please add $5 for shipping within the U.S., $30 overseas. $_____

■ Total. $_____

MICROSOFT WINDOWS

The Whitewater Group®

Technology Innovation Center
906 University Place, Evanston, IL 60201

Circle **no. 282** on reader service card.

Four PROLOGs for the Macintosh

by Dan L. Pierson

The growth in popularity of PROLOG has been vividly reflected in the products available for the Macintosh. Four PROLOGs have been released since January 1986; others have been announced or are rumored. This review covers the four PROLOGs that were available for the Mac as of November 1986: PROLOG/m, Version 1.10a; AAIS Prolog M-1.10; MacPROLOG, Version 1.0a; and ExperProlog-II, Version 2.32.

The Products

PROLOG/m

PROLOG/m from Chalcedony Software is a basic Edinburgh PROLOG interpreter. It and the IBM PC version, PROLOG/i, are the successors to the several-year-old PROLOG/V. The package includes a 184-page paperback *Language Reference and Tutorial*; a 30-page paperback manual; and a bootable, unprotected disk that contains the PROLOG/m program, the PROLOG library, a file of release notes, and a folder of example programs. Chalcedony also sells three packages of extension programs: the TOOLBOX, the TOYBOX, and NFL-Xpert. This review includes the TOOLBOX, a collection of 58 utility programs. Some of the TOOLBOX programs, such as setof and bagof, are standard parts of other PROLOGs. Other TOOLBOX programs, such as avl_tree and h_list, are useful tools that are not part of any other PROLOG. The TOOLBOX consists of a 72-page paperback book that contains the complete source of all the programs, explanations, and examples of use and a disk containing the program sources.

Dan L. Pierson, 10 Fort Meadow Dr., Hudson, MA 01749. Dan was one of the developers of the first release of VAX Common LISP. He is currently developing software for Unix.

It seems that PROLOG's implementors never considered syntactic similarity to be a goal.

AAIS Prolog

AAIS Prolog is the newest of the products. The first shipments for the Mac were in September 1986 (there was an earlier Unix version), and the first update was shipped in late November. AAIS is a large, elaborate Edinburgh-based system. The AAIS Prolog package includes a 134-page, letter-size manual; the primer *Prolog Programming for Artificial Intelligence* by Ivan Bratko; a bootable, unprotected disk containing the PROLOG program and library; and a disk with a few example programs.

The November update to AAIS Prolog was supposed to include support for Macintosh graphics. Much to my surprise this support took the form of general, user-extendable support for both the entire Macintosh ROM and any additional programs in code resources. This means that anyone with access to an assembler or compiler that can produce code resources can extend AAIS Prolog simply by adding the resource to a copy of the PROLOG file using Apple's ResEdit and writing a small PROLOG program to load the resource and define the new predicates. Of course, this is not novice programming—the price for AAIS Prolog's flexibility is that graphics and window programming are much more complicated than in a more limited, higher-level environment.

MacPROLOG

MacPROLOG is micro-PROLOG, the first dialect of PROLOG for microcomputers. Micro-PROLOG and its big sister sigma-PROLOG are currently available for MS-DOS, CP/M-86, CP/M-80, the Apple II, the Commodore 64, most Unix machines, VAX/VMS, and Data General machines running AOS. MacPROLOG is a complete, professional development system with many special features. The MacPROLOG package includes a PC-style ring binder containing a 61-page users' guide and a large reference manual; the primer *micro-PROLOG* by Clark and McCabe; an unbootable, unprotected disk containing the PROLOG program, the Generic (all three syntaxes) programming environment, a file of release notes in two formats (MacWrite and text), and a folder of example programs; a disk containing the Standard and Edinburgh programming environments; and a disk containing the Standard and Edinburgh run-time systems.

Two of the most impressive features of MacPROLOG are its support for three syntaxes with automatic conversion between them and its extensive support for the Mac interface. Predicates exist to perform simply and easily almost any task involving windows, menus, and dialogs. For example:

```
(SCROLL_MENU
  ["please select some" fruit]
  [apples pears peaches]
  [apples] _s)
```

displays a moded dialog box containing the prompt "please select some

fruit," a scrolling list of the fruit with apples highlighted, and an OK button. When the user clicks OK, the variable _s is unified with the selected fruit. Using the above facilities, a run-time system, and user-defined error handlers, an application can hide the underlying PROLOG completely. Given all this power, it's surprising that MacPROLOG provides absolutely no support for Macintosh graphics.

ExperProlog-II

ExperProlog-II is a polished, professional production of Alain Colmerauer's next-generation PROLOG. ExperTelligence advertises compatible versions for VAX/VMS and the IBM PC; the reference manual also mentions Hewlet-Packard (HP-150, HP-1000, HP-9000) and expands the IBM PC to include all MS-DOS machines. The package includes an attractive vinyl ring binder containing a 171-page reference manual and 132-page Macintosh users' manual; the primer *PROLOG* by Francis Giannesini et al.; a bootable, unprotected disk containing the PROLOG program, an initial saved system, and a demo program; and a disk containing example programs and the Lisa Pascal programs, object files, and link commands for a sample extension to Prolog-II.

ExperProlog-II provides extensive, fairly high-level support for Macintosh graphics and mouse input but very little menu support (extend one menu) and no support for dialogs or applications that hide the underlying PROLOG. User extensions to PROLOG could probably do most of this, but extensions currently require the Lisa Workshop—an uncommon, expensive, and obsolete development environment for the Mac.

User Interfaces

All four PROLOGs provide a basic Macintosh environment with a menu bar, desk accessory support, and an initial window. Table 1, right, summarizes the user interface features of the four products. The main interaction window is called a query window because you interact with a PROLOG system by asking it questions. The first three entries in the table refer to features of this query window. The next section refers to the program editor, if any. Only the AAIS editor is suitable for editing non-PROLOG text—the other editors are too closely tied to the PROLOG evaluation mechanism. MacPROLOG includes a Find Definition menu option that makes the correct window visible and positions the cursor at the start of the requested definition. Most of the PROLOGs report errors by a combination of an alert dialog, positioning the cursor, and highlighting the erroneous text. The method for interrupting a long running program varies from none in Prolog-II to using the Mac's Programmer's Switch to invoke a full debugger breakpoint in AAIS Prolog.

Debugger

All the PROLOGs provide some sort of debugger. The traditional PROLOG debugger is called a box debugger because it is based on viewing a PROLOG procedure as a box with two entries—call and redo—and two exits—fail and exit. With a full box debugger, tracing and breakpoints (called spy points) are individually controlled for each entry and exit in the box for each procedure. Individual control of the entries and exits is done by leashing or unleashing the interpreter for each type of entry or exit. PROLOG/m provides full leashing and unleashing but few other commands. Prolog-II provides only an uncontrolled trace. The other two PROLOGs provide box debuggers without full leashing; the AAIS Prolog debugger provides a particularly rich selection of other options.

The last row in Table 1 shows how much free memory remained on a 512K Macintosh after starting the basic, fully loaded system. I couldn't determine this number for PROLOG/m. The other three PROLOGs provide options to remove features and save some memory. I didn't have time to reconfigure each PROLOG to see how much memory can be saved if all the optional features are stripped out.

The Dialects of PROLOG

PROLOG is more than merely another language that has never suffered the pangs of standardization; it is a language whose primary implementors seem never to have considered syntactic similarity, let alone compatability, a goal. The four PROLOGs in this review support five distinct syntaxes, only one of which is merely a significant extension of another.

The smallest elements of a PROLOG

	PROLOG/m	AAIS Prolog	MacPROLOG	Prolog-II
Queries				
Scrolling	no	yes	dialogs[1]	yes
Editable	yes	yes	yes	yes[2]
First/all	choice[3]	pause[4]	choice[5]	all
Editor	no[6]	yes	yes	yes
Multifile	—	yes	yes	no
Can query	—	yes	yes	no
Search	—	yes	yes	yes
Replace	—	yes	yes	no
Balance parens	—	no	option	auto
Find definition	—	no	yes	no
Errors				
Report	message	dialogs	dialogs	dialogs
Cursor	no	yes	yes	yes
Highlight	no	yes[7]	yes	yes
Interrupt	command-	Programmer's Switch	Programmer's Switch	no[8]
Debugger	box	box[9]	box[10]	trace
Free memory (bytes)	?	84,524	81,850	78,960

1. Multiple query dialogs can be open at a time; there is a default scrolling output window.
2. Special Edit Goals window.
3. Global menu/command choice.
4. Pause after each solution; type ';' for next solution.
5. Elaborate choices in each query dialog.
6. Can use an editor desk accessory.
7. Highlight removed when dialog is exited; dialog sometimes covers highlight.
8. Programmer's Switch interrupt is recoverable but leaves the screen messed up and rudely aborts the execution state.
9. No leash/unleash options but lots of stack display and execution modification commands.
10. Only interpreted predicates can be traced; all predicates can have spy points set on them.

***Table 1:** User interface*

(continued from page 31)

program are clauses. A clause consists of a functor and zero or more arguments. The number of arguments of a clause is the arity of the clause. A clause of zero arity is an atom. Clauses are the elements of rules. A rule consists of a head (the left-hand side) and a body (the right-hand side). The head of a rule consists of a single clause; the body consists of zero or more clauses. The head of a rule is true if and only if all the clauses in the body are true. A rule with no body is always true and is known as a fact. A set of rules whose heads all have the same functor and arity form a procedure. A procedure defines the meaning of a particular functor and arity, called a predicate. Because the same functor can have different meanings with different arities, predicates are typically referred to by both functor and arity, as in *plus/3* for plus with three arguments.

Edinburgh Syntax

The most common PROLOG syntax is called Edinburgh syntax after the Prolog-10 and C-Prolog systems developed at the University of Edinburgh. This syntax is used in the well-known PROLOG primer *Programming in Prolog*, the excellent new intermediate book *The Art of Prolog*, and many other books and papers on the language. Edinburgh is the only syntax that supports a wide selection of infix operators. All of the products except Prolog-II support the Edinburgh syntax, either natively or as an option.

A typical Edinburgh procedure looks like this:

```
reverse([],[]).
reverse([X|Xs],Zs) :-
   reverse(Xs,Ys),
   append(Ys,[X],Zs).
```

This is the famous "naive reverse" procedure for reversing a list. It reads: the null list is its own reverse; to reverse a nonnull list, concatenate the reverse of the tail of the list and the head of the list.

In Edinburgh syntax, clauses are expressed as *functor(arg[, . . .])*, and rules are *clause :- clause [,clause . . .]*, where the commas between clauses mean *and*. Lists are expressed as comma-separated terms within square brackets; the head and tail of a list are indicated by a vertical bar. Thus *[X|Xs]* means the list with head *X* and tail *Xs*. Symbols start with a lowercase letter, whereas variables start with an uppercase letter. Arithmetic expressions are expressed as *X1 is X + 1*. Note that *X* and *X1* are different variables; once a PROLOG variable is bound to a value, it cannot be changed. Operators such as *is* and + can be freely defined with arbitrary precedence, associativity, and meaning. There are two types of comments: end-of-line comments starting with *'%'* and comment text bracketed by nonnesting *'/*'* and *'*/'* as in C. MacPROLOG's Edinburgh mode does not permit the end-of-line comment syntax.

AAIS Prolog's native syntax is an extension of Edinburgh syntax that supports AAIS' new features and data types. AAIS Prolog is about as similar to Edinburgh PROLOG as Common LISP is to its ancestor MacLisp.

The naive reverse procedure is exactly the same in AAIS and Edinburgh PROLOGs. Among the syntactic changes in AAIS Prolog are that *'c'* is a character instead of a small integer (but can automatically be treated as an integer when needed), *cat* is a string instead of a list of small integers, and *foo:append* is the symbol *append* in the package *foo*. Also, packages are a type of module; programs in one package will not see symbols of another package unless the program's package inherits from the other package.

Standard Syntax

MacPROLOG's native (Standard) syntax is LISP-like; everything is fully parenthesized and stored as lists. The naive reverse procedure is expressed in Standard syntax as:

```
((reverse ( ) ( )))
((reverse (_X|_Xs) _Zs)
   (reverse _Xs _Ys)
   (APPEND _Ys (_X) _Zs))
```

The History of PROLOG

Interest in using formal logic as a programming language dates back to research in automatic theorem provers in the early 1950s. Robinson's 1965 paper[1] provided the necessary groundwork for a practical logic programming language. Hewitt's PLANNER,[2] although later recognized as a failure, was the first logic-based programming system. Cooperative research by Alain Colmerauer and Robert Kowalski resulted in the creation of the first PROLOG interpreter in the early 1970s.

PROLOG research and development continued in Europe during the 1970s. The two principle research groups were Colmerauer's group at the University of Marseille-Aix and the University of Edinburgh group, which included Robert Kowalski and David Warren. Warren was responsible for the next major breakthough in PROLOG. His Prolog-10 compiler, the first high-performance PROLOG system, did much to dispel the belief that logic programming languages had to be horribly inefficient.

The Japanese gave PROLOG its next big boost when they decided to use PROLOG instead of LISP as the basis of their Fifth Generation Project.

Though still controversial (see Carl Hewitt's attack in the premier issue of *AI Expert* magazine[3]), PROLOG is gaining force as the second major language for AI applications. Serious implementations of all the major dialects are available for most computers and operating systems from micros to mainframes. New language features are appearing to address many of the perceived shortcomings of basic PROLOG. Whether you believe in the virtues of PROLOG or not, the language is maturing as a real, long-term option in the programming world.

PROLOG is an important language to understand because it requires a new way of thinking about programming problems. PROLOG is a declarative, side-effect-free language with a predefined processing loop. The absence of side effects in "pure" PROLOG makes the language a serious candidate for parallel processing. Indeed, several concurrent languages based on PROLOG, such as Concurrent Prolog,[4,5] have already appeared.

The Power of DOS is Yours!

On Command: Writing A Unix-Like Shell for MS-DOS

by Allen Holub

This book and ready-to-use program demonstrate how to write a Unix-like shell for MS-DOS. **On Command** includes an enhanced, working version of Holub's popular Unix-like shell, along with a detailed description of the Shell and complete C source code.

The techniques you'll learn are applicable not only to MS-DOS, but to most other programming environments as well.

You'll find how to do interpretive control flow in any C program, a thorough discussion of low-level DOS interfacing and significant examples of C programming at the system level.

The Shell's supported features include: read, editing, aliases, history, redirection and pipes, Unix-like command syntax, DOS- compatible prompt support, and C-Shell-based shell scripts. A new Shell variable expands to the contents of a file so a program can produce text that is used by Shell scripts.

The Unix-like control flow includes: if/then/else; while; foreach; switch/case; break; continue.

The ready-to-use program and all C source code are included on disk. The Shell works on IBM PC's and compatibles.

/Util When used with the shell, this collection of utility programs and subroutines provide you with a fully functional subset of the Unix environment! Utilities include: cat; cp; date; du; echo; grep; ls; mkdir; mv; p; pause; printevn; rm; rmdir; sub; and chmod. Complete source code and manual are included.

Receive **On Command** together with **/Util** for only $59.95!

On Command	**Item #163**	**$39.95**
/util	**Item #161**	**$29.95**
On Command with /Util	**Item #164**	**$59.95**

Taming MS-DOS

by Thom Hogan

Learn how to make DOS work for you! Taming MS-DOS takes you beyond the basics, picking up where your DOS manual leaves off.

- Learn to maximize your batch files with routines using redirection, filters and pipes. You'll find routines that prevent accidental reformatting of your hard disk, redefine function keys and locate files within subdirectories. You'll learn to implement a DOS help system with help text files, a menu system that interprets keyboard input, and a routine for quick redefinition of function keys.
- Learn to customize CONFIG.SYS to maximize the performance of your system and how to use ANSI.SYS to tailor your system prompt and monitor attributes to fit your needs.
- Taming MS-DOS includes nearly 50 ready-to-use programs that increase DOS's functionality. Now you can easily rename directories and disk volumes, change file attributes, check available RAM and disk memory, display a memory resident clock, and assign DOS commands to ALT keys.
- Quick reference charts provide easy access to batch command syntax, CONFIG.SYS syntax and ANSI.SYS command strings.

All programs, including batch files and DOS enhancements, are available on disk with full source code.

Taming MS-DOS	**Item #060**	**$19.95**
Taming MS-DOS with disk	**Item #061**	**$34.95**

Return form
OR

CALL
800-533-4372
TOLL FREE

In CA call
800-356-2002

YES!
Please send me:

Item #060 Taming MS-DOS $19.95	________
Item #061 Taming MS-DOS with disk $34.95	________
Item #163 On Command $39.95	________
Item #161 /Util $29.95	________
Item #164 On Command & /Util $59.95	________
Item #090 SK: The System Kernel $49.95	________
Item #091 DS: The Display Driver $39.95	________
Subtotal	________
CA Residents add tax	____ % ____
Add $2.25 per item for shipping	________
TOTAL	________

☐ Check Enclosed. **Make Payable to M&T Publishing, Inc.**

Charge my ☐ VISA ☐ M/C ☐ Amer. Exp

Card # ______________________ Exp. ________

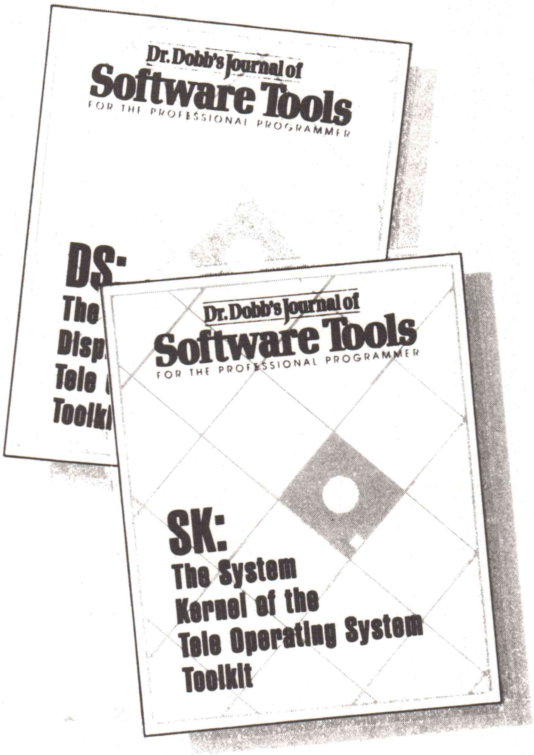
Name ______________________

Address ______________________

City ______________ State ________ Zip ________

3126A

The Tele Operating System Toolkit



The unique features of this four part, multitasking operating system will allow you to fully exploit the power of any 8086-based machine! **Tele** is written in C and assembly language for IBM PC compatibles and includes **preemptive multitasking capabilities and an unlimited number of tasks.** Tele contains **full C and Assembler source code,** as well as precompiled libraries. It is compatible with MS-DOS, Unix, and the MOSI standard. MS-DOS disk format.

SK: The System Kernel includes the most crucial part of the Tele Operating System - the preemptive multitasking algorithm. **SK** also contains an initialization module, general purpose utility functions for string and character handling, format conversion, terminal support and machine interface, along with a real-time task management system. All other components require SK: The System Kernel.

SK	**Item #090**	**$49.95**

DS: The Display Driver contains BIOS level drivers for a memory- mapped display (the fastest way to display data), window management support and communication coordination between the operator and tasks in a multitasking environment. **DS** includes functions to create and delete virtual displays, and functions to overlay a portion of a virtual display on the physical display.

An unlimited number of virtual displays can belong to any particular task, and an unlimited number can be in the system at any time. Requires SK: The System Kernel.

DS	**Item #091**	**$39.95**

CALL
800-533-4372
TOLL FREE

In CA call
800-356-2002
OR
Return form

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

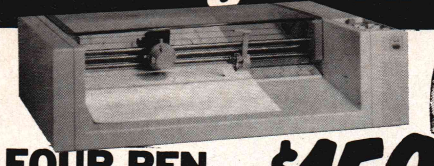
FIRST CLASS PERMIT 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

M&T Books
501 Galveston Dr.
Redwood City, CA 94063

California Digital

17700 Figueroa Street • Carson, California 90248



FOUR PEN COLOR PLOTTER $159

The manufacturer has asked us not to publish their name. But this four color plotter was produced by one of the Worlds largest makers of personal computers.

The 410 color plotter will connect to the **serial port** of virtually any micro-computer. Simple ASCII commands direct one of the four color pens to draw circles, arcs or ellipses on paper or transparency material up to 11 by 17 inches. The plotter is capable of producing the full upper and lower case alphabet along with seven international character sets. Text can be printed horizotal, vertical or diagonal in sizes from 1/16 to 6 inches, slanted forward or backward to 85 degrees.

Enlargements or reductions are achieved through elaborate firmware. Pen travel is four inches per second with .004" pen resolution. Standard pens are available in an assortment of 32 different colors and widths.

The ideal plotter for architecture, CAD engineering or graphic design. **At $595 it was a great buy, at $159 its a steal.** Support packages for specific computers available. Manual only $15 refundable upon purchase of plotter.

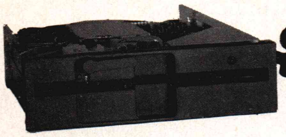
OTHER PLOTTERS AVAILABLE:
Hewlett Packard • HoustonInstruments • Roland
Sweet P • CalComp... please call for prices.

Qume 142 PC Compatible

$65
Quantity Two

These Qume double sided PC compatible drives bear the IBM logo. The drive was manufactured for IBM for use in the recently discontinued PC Junior computer. We have purchased a large quantity of these drives and are currently offering them in sets of two for only $65 each.

Panasonic AT Compatible

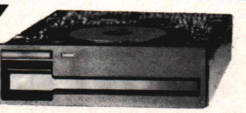


$115

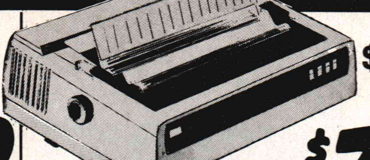
The Panasonic 475 is jumper selectable between 96 TPI format and 1.2 Megabyte as used in the IBM/AT computer.
Suggested price of this drive is $189 but because of unusual buying opportunity, California Digital is able to offer these drives at only $115 each, quantity two.

	One	Two	Ten
QUME 142 half height	69	65	call
TEAC FD55BV half height	109	99	89
TEAC FD55FV 96 TPI, half ht.	119	109	105
TEAC FD55GF for IBM AT	169	159	155
PANASONIC 455 Half Height	109	99	89
PANASONIC 475 1.2 Meg./96	119	115	109
MITSUBISHI new 501 half ht.	129	119	109
MITSUBISHI 504A AT comp.	169	159	155
Switching power supply			49
Installation Kit with manual			10
Dual enclosure for 5¼" drives			59
34 pin edge connectors			5
Scotch head cleaning kit			19
Flip & File Storage tubs			15

SONY $139



The Sony 53W is a 3½ inch double sided double density disk drive. The drive can be connected to your existing floppy controller but will require MS/DOS 3.2 to properly operate in the PC and compatibles. The 53W is similar to the drive being used in the PC portable.



$2495

$759 80 Character Daisy Wheel Printer

These Fujitsu Daisy Max 830 were manufactured for Motorola's Computer Division. The purchase order was canceled and Fujitsu was forced to liquidate these 80 character per second daisy wheel printers at "fire sale" prices.

Features: bullet proof construction, serial RS-232 interface, Diablo 630 wheels and commands, programmable line spacing in increments of 1/96" and column spacing of 1/120". The printer is also capable of underscoring, bold overprint, shadow print, centers and justifies along with vector plotting.

Factory suggested price of the Daisy Max 830 was $2495, while supplies last California Digital is offering this liquidated special at only $759. Also available: tractor and sheet feeders.

Bernoulli Box

was $3540 now only
$1595

The **Bernoulli Box** by Iomega, features 10 and 20 megabyte removable cartridges, and delivers reliability, expandability, transportability, security and speed in one versatile subsystem. It lets you transfer megabytes of information safely and swiftly for primary or backup storage. Or combine several software programs onto a single cartridge for easy switching from one to another.

Reliable... the Box has incredible resistance to shock and vibration completely eliminating the possility of head crash.

Expandable... grow at your own pace by adding inexpensive cartridges. When **security** is essential, don't lock up your system... just lock up the cartridges.

The Bernoulli Box delivers preformance that often exceeds the best of hard disk speed and the convenience of floppy disks. At these prices don't be caught wishing you had one after a loss of irreplacable data.

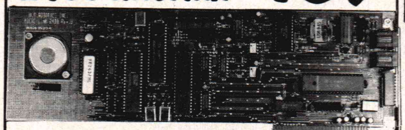
		List	Our Price
10+10 Meg.	**A2210H**	**$3450**	**1595**
20+20 Meg.	**A2220H**	**4540**	**2095**
Bootable Controller		**255**	**189**
10 Meg. Cartridge		**79**	**49**
20 Meg. Cartridge		**99**	**65**

Smarteam 2400 Baud

$289

The Smarteam 2400 offers all the features of the Hayes Smart Modem 2400 for a fraction of the price. Now is your opportunity to purchase a 2400 baud modem for only $289.
Also available: The Smarteam 1200 at only $139

U.S. Robotics 2400 Internal $189



The US Robotics Micro 2400 modem is one hundred percent Hayes compatible, auto dial, auto answer, auto everthing. A super value at only $189.

$99

This 300/1200 baud modem matches design specs of Bell 212A and for all practical purpose those of the Hayes Smart Modem. The Avatex brings you reliable data transmission for only $99.

OTHER MODEMS AVAILABLE:
Hayes • Universal Data • Fujitsu
Prometheus • Anchor • US Robotics

NEW $1895

HEWLETT PACKARD

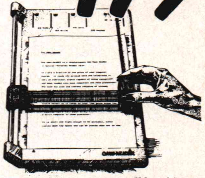
New from Hewlett Packard the Series II LaserJet Plus. More features than the LaserJet but at a much lower price. Smaller footprint (not as pictured), 512 K/byte memory expandable to 4 Megabytes. List price $2495 California Digital price $1895.

Other Laser Printers available:
Texas Instruments • Okidata • Apple • Ricoh • Qume
AST Research • QMS • Xerox • Quadram • Centronics

NOW YOUR COMPUTER CAN READ!!

Omni-Reader... the first optical character reader **designed and priced for the small computer**

$179

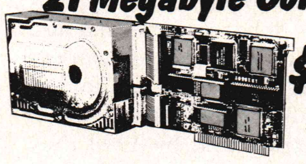


COPY:
•Manuscripts •Contracts •Articles
•Forms •Invoices

APPLICATIONS:
•Mailing Lists •Editing •Data Base Management •Transfering information between incompatible systems

Uses a standard RS-232 serial port hookup to interface eaisily with your computer.

21 Megabyte Gold Card



$479

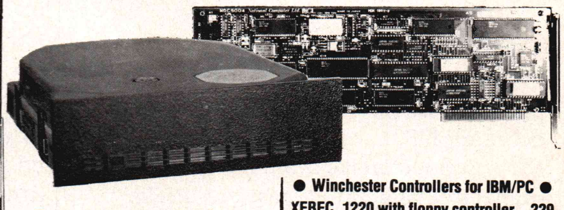
Also available 31 Megabytes at only $559.

The fastest, lowest powered, longest warrantied, most durable, highest capacity, most reliable, lowest priced HARD-DISK-ON-A CARD available in the world today.

• 65ms Access Time - *Fastest Available* • Automatic Head Unloading - *Protects Heads and Media* • 2K Sector Buffer - *Increases System Throughput* • High Reliability: 28,000 hr. MTBF *No one else even close* • 15.5 Watt Power Consumption - *Lowest Available* • 2 Year Warranty *Longest Available*

Seagate $459

30 MEGABYTE WINCHESTER HARD DISK KIT



Five Inch Winchester Disk Drives

	each	two+
SEAGATE 225 20 Meg. ½ Ht.	329	299
SEAGATE 238 30 Meg. RLL	389	359
SEAGATE 4026 26 M. 35mS.	659	629
SEAGATE 4051 51 M. 35mS.	795	759
FUJITSU 2242 55 M. 35mS.	1399	1329
FUJITSU 2243 86 M. 35mS.	1895	1819
RODIME RO-202E 27 Meg.	659	629
RODIME RO-203E 40 Meg.	995	959
RODIME RO-204E 53 Meg.	995	959
CONTROL DATA 94155-86 M.	1829	1779
MAXTOR XT1140 140 Meg.	2595	2529
TOSHIBA MK56 70 M. 30mS.	1789	1729
TANDON 502 10 Meg.	419	379

• Winchester Controllers for IBM/PC •	
XEBEC 1220 with floppy controller	229
DTC 5150CX	139
OMTI 5520 half card	129
OMTI 5527 RLL controller	159
ADAPTEC 2070 RLL controller	179
ADAPTEC 2010A	159
WESTERN DIGITAL WD/1002WX2	119
• SCSI/SASI Winchester Controllers •	
XEBEC 1410A 5¼" foot print	219
WESTERN DIGITAL 1002-05E 5¼"	289
OMTI 20L	119
• Winchester Accessories •	
Installation Kit with manual	10
Winchester enclosure and supply	139
Dual 20/34 cable set	25
Switching power supply	49

VISA
master charge THE INTERBANK CARD

Shipping: First five pounds $3.00, each additional pound $.50.
Foreign orders: 10% shipping, excess will be refunded.
California residents add 6½% sales tax. • COD's discouraged.
Open accounts extended to state supported educational institutions and companies with a strong "Dun & Bradstreet" rating.

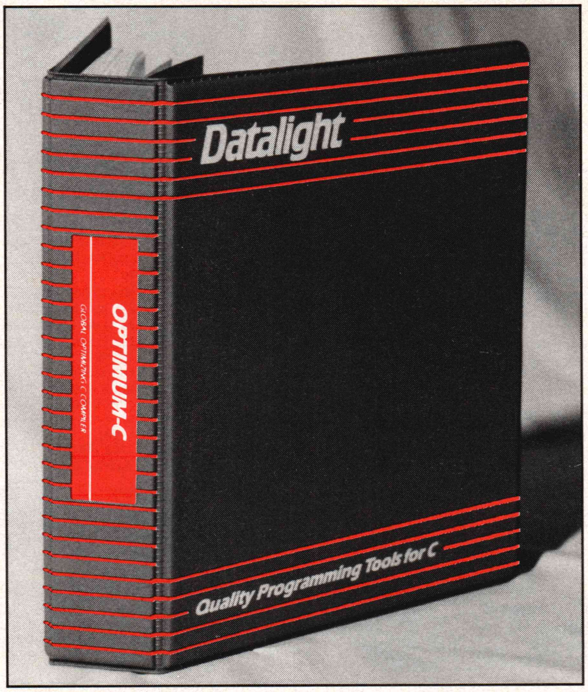
TOLL FREE ORDER LINE
(800) 421-5041
TECHNICAL & CALIFORNIA
(213) 217-0500

A Challenge to Microsoft® C...

We challenge Microsoft C (Ver 4.0) to a C compiler duel to the finish, measuring compile, link, and execution times. If they win, we will stop advertising for two months.

by Roy Sherrill

If Microsoft C (Ver 4.0) can beat our new C compiler, Optimum-C then we will stop advertising in all magazines and publications for two full months and win or lose we will publish the results in its entirety. Even the Microsoft ads say "The Fastest C you've ever seen," so let the challenge begin.



Optimum-C challenges Microsoft C Version 4.0 to a duel, yet is priced at only $139 (includes free learn C tutorial) which is one-third suggested retail price of Microsoft C Version 4.0

Walter says Optimum-C is better

It all started when Walter Bright, the developer of Optimum-C, was explaining his new global optimizing C compiler and how it's code would be faster than Microsoft C (Ver 4.0). Walter and I were frustrated because here we had a C compiler that would beat Microsoft C on 7 out of 10 benchmarks and also compile and link faster; yet our marketing consultant, Mark Astengo, told us that Microsoft C had a lock on the C compiler market and by 1990 they would probably have an 80% market share. Then Mark said, "Roy, if your C compiler is as fast as you say it is, why not challenge Microsoft C to a duel? Furthermore, if Microsoft wins, Datalight should stop advertising for two months and print the results of the test, win or lose." Well, I've always been one for a challenge. So here it is...

We only ask the following...

The benchmark suite will consist of the set of programs that Microsoft supplied to *Computer Language* for their February 1987 C compiler review issue. Microsoft will make available the programs to Datalight at least two weeks prior to the benchmarking. The benchmarking will be between Microsoft C 4.0 and Optimum-C. It will occur at a mutually agreed upon time and place. Interested individuals will be allowed to attend. The benchmarks will be compiled and run on a standard IBM PC-AT.

There will be two separate tests for each program: compile and link speed, and execution speed. For each test, a representative from each company will set up the compiler so that it performs at its best.

The benchmarks will be adjusted so that they take sufficiently long to run, that the tolerance involved in timing them is insignificant. The winner is determined by the compiler with the faster execution times for the majority of the benchmarks. We'd like an answer from Microsoft no later than April 1, 1987.

So what's a global optimizer?

A global optimizer looks at an entire function at once, analyzing and optimizing the whole function. A technique called data flow analysis is used by Optimum-C to gather information about each function. This enables your compute-bound programs to execute as much as 30% faster after global optimization. But there is one catch... because the global optimizer ruthlessly searches for ways to speed-up execution speed and minimize memory usage, it has relatively slow compile times. No need to worry, though, because you can merely turn the global optimizer off. In fact, you can select which optimizations are desired. Listed on the next page are 12 optimizations. It's your choice: all, none, or partial...

Optimum-C Version 3.0

- Full UNIX System 5 C language plus ANSI extensions
- Fast/tight code via powerful optimizations including common sub-expression elimination
- DLC one-step compile/link program
- Multiple memory model support
- UNIX compatible library with PC functions
- Compatible with DOS linker and assembler
- Third-party library support
- Automatic generation of .COM files
- Supports DOS pathnames, wild cards, and Input/Output redirection
- Compatible with Lattice C version 2.x
- Interrupt handling in C
- Debugger support
- ROMable code support/start-up source

MS-DOS® Support Features

- Mouse support
- Sound support
- Fast screen I/O
- Interrupt handler

MAKE Maintenance Utility

- Macro definition support
- MS-DOS internal commands
- Inference rule support
- TOUCH date manager

Tools in Source Code

- cat—UNIX style "type"
- diff—Text file differences
- fgrep—fast text search
- pr—Page printer
- pwd—Print working directory
- wc—Word count

1. Constant propagation
2. Copy propagation
3. Dead assignment elimination
4. Dead variable elimination
5. Dead code elimination
6. Do register optimizations
7. Global common subexpression elimination
8. Loop invariant removal
9. Loop induction variables
10. Optimize for space
11. Optimize for time
12. Very busy expressions

Quite frankly, if it were not for the global optimizer we would be much more timid in regards to our Microsoft challenge. So, if you want minimum compile and link times with good code execution speed, then compile without the optimizer; and if you want the fastest possible execution time, use our global optimizer.

Choose from five memory models

Speed your programs by selecting the memory model that best suits your application.

Memory Models

Model	Code	Data
Compact	64k total code & data	
Small	64k	64k
Program	1M	64k
Data	64k	1M
Large	1M	1M

Compiling, one step...

Now with the one step DLC program you can create .OBJ, .EXE and .COM files. Also, DLC can handle multiple files and run MASM on your assembly files.

Includes complete source code for library

The UNIX-compatible library includes complete source code. Experienced programmers can use the source code to configure and rebuild the library to suit the application.

Third party library support includes Light Tools (Blaise), ZVIEW (Data Management Consultants), MASM (Microsoft), JACK (Cracker Jack), APL2C (Lauer Software), BTREE, ISAM (SoftFocus), Vitamin C (Creative Programming), DSD (Soft Advances), with more on the way.

One concise manual says it all in under 300 pages

If you are tired of wading through reams of documentation, worry not, because Optimum-C includes one complete and concise programmer's manual. The manual is delivered in an IBM-style 3-ring binder which includes nine chapters, appendices, and easy-to- follow examples.

Dr. DOBBS and COMPUTER LANGUAGE Agree

Even though they reviewed our compiler before it had our global optimizer, both *Dr. Dobbs* and *Computer Language* were surprised by Datalight's performance. Read what they said...

"Reviewing this compiler was quite a surprise for us. For such a low price, we were expecting a "lightweight" compiler. What we got was a package that is as good as or better than most of the "heavyweights." Datalight C implements a complete C language. It also compiles quickly, doesn't take up much disk space, and looks impressive in the benchmarks."

DR. DOBBS*, *August 1986

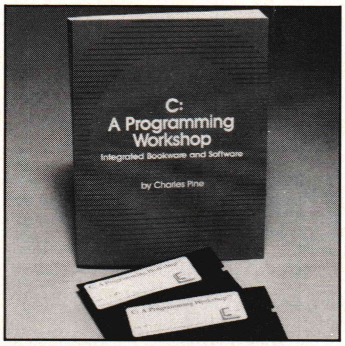
"This is a sharp compiler!... what is impressive is that Datalight not only stole the compile time show completely, but had the fastest Fibonacci executable time and had excellent object file sizes to boot!"

COMPUTER LANGUAGE,
February 1986

Switch to C and we'll help!

Even if you have never programmed in C before, you'll find the C tutorial included free with Optimum-C a real time saver. If you don't have need for a C tutorial give this to a friend. (It's a $49 value) This combination workbook and floppy disk will lead you through the C language with tutorials, quizzes, and program exercises.

*FREE C TUTORIAL!



"Combines print and software technology to create the most integrated new type of training system I have ever seen."

ADAM GREEN, INFO WORLD
January 27, 1986

Circle **no. 203** on reader service card.

Try Optimum-C risk free

Load this C compiler in your microcomputer and write a C program. Then notice that you don't have to wait as long while compiling and linking. Also notice the fast execution times, or if you haven't programmed before, use the free C tutorial to walk you through the land of C.

If you are not 100% satisfied, simply return Optimum-C in it's original carton within 30 days for a full refund.

To order Optimum-C risk free with your credit card, call toll free or send your check for only $139 (plus shipping and handling).

O.K. Microsoft, it's up to you. We've put two months of advertising on the line that says you can't beat Optimum-C to a real test. Your answer, please?

PRICES

Developer's Kit	still only $99
Optimum-C	$139

Add $5 for shipping in US/$15 outside US
COD (add $2.50)

Not Copy Protected

CALL TOLL-FREE TODAY!
1-800-221-6630

In Washington and Outside U.S. call...
206-367-1803

ATTENTION!
OEM DEVELOPERS
LARGE CORPORATE ACCOUNTS

We have developed an excellent OEM program that can be tailored to your needs. OEMs may:
1. Get OEM discounts
2. Buy partial reproduction rights
3. Buy complete reproduction rights

Also, corporate discounts are available to volume users. Please contact me directly to discuss your specific needs.

Roy Sherrill
Roy Sherrill
President

Datalight

Box 82441
Kenmore, Washington 98028
(206) 367-1803

*Limited offer available exclusively to readers who purchase directly from Datalight.

MAC PROLOGS
(continued from page 32)

Clauses are expressed as *(functor arg [arg . . .])* and rules are *(clause [clause . . .])*. Lists are a parenthesized sequence of terms, and a rule is a list. As in Edinburgh syntax, a vertical bar separates the head and tail of a list. Symbols start with any alphabetic character. Variables are distinguished by a leading underscore. Arithmetic expressions are expressed as *(+ _x 1 _x1)*; infix operators are not available. Multi-element structures with compact storage similar to Edinburgh clauses are available as tuples expressed as <a_b1>; special functions are provided to compose and access tuples. Comments are bracketed with nonnesting '/*' and '*/'. This syntax is alleged to make it much easier to write programs that manipulate other programs.

Simple Syntax
MacPROLOG also supports Simple, an English-like syntax for beginning PROLOG students. It is used in several primers, including *micro-PROLOG*. The naive reverse procedure in Simple syntax is:

```
(_X) reverse (_X)
(_X _Y|_Xs) reverse _Zs
if(_Y|_Xs) reverse _Ys
  and append (_Ys (_X) _Zs)
```

Clauses are expressed as *functor (arg [arg . . .])* or *arg functor* or *arg1 functor arg2*. Rules are built of clauses, conditions, and conjunctions. In the preceding example, *and* is a conjunction and *if* is a condition. Lists, symbols, variables, and comments are the same as in Standard syntax. Simple arithmetic expressions are as in Standard syntax, but complex expressions such as:

```
(SUM (X + Y / 5) Z X1)
```

are possible.

A New Model
The naive reverse procedure in Prolog-II is:

```
reverse(nil, nil) -> ;
reverse(X.X-tail, Z) ->
  reverse(X-tail, Y)
  append(Y, X.nil, Z);
```

Clauses are still expressed as *functor(arg[,arg . . .])*, but rules are now *clause −> clause[clause];*. The *−> ;* is required even for a fact. Lists are expressed as a series of terms separated by '.' and ending in *nil*. List head and tail are expressed as *X.X-tail*. Symbols start with at least two alphabetic characters. The rest of the symbol can contain dashes but not underscores. Several extended characters in the Mac's character set are treated as alphabetic, so symbols such as *Ægis* are legal. Variables start with one alphabetic character, optionally followed by any number of digits, optionally followed by any number of apostrophes, optionally followed by a dash and any symbol. This allows such names as *x*, *x1*, *X''*, and *X-the-first* but disallows such mnemonic names as *Premise* and *Denial*. Arithmetic expressions are expressed as *val(add(X,1),X1)*. Comments are strings surrounded by ". Comments cannot span lines and cannot appear within the rules defining a single function (for example, an

end-of-line comment after the first rule of naive reverse would be illegal). Program lines indented with tab characters cause syntax errors, such as "A SIMPLE TERM IS EXPECTED," with the cursor positioned near the tab but nothing highlighted.

Prolog-II terminology can confuse programmers who are used to earlier versions of PROLOG. Where other PROLOGs refer to "rules" and "unification," Prolog-II talks about "trees" and "deletion." There is a reason for the change in terminology. Prolog-II is based on a new, expanded, theoretical model, which adds the concept of inequalities between trees to the earlier PROLOGs' equalities. This feature, encapsulated in the built-in rule *dif*, adds a great deal of power to Prolog-II.

Prolog-II's other new features include infinite trees and error handling. Infinite trees, which consist of otherwise finite trees with loops, provide direct PROLOG support for the directed graph structures found in such applications as finite-state automata and grammars. Error handling in Prolog-II is based on the primitive *block*, which provides a simple form of signal handling. All Prolog-II's standard run-time errors signal with *block-exit* so that a program can handle the error. Previous PROLOGs either exit fatally on error or treat errors as failure; neither approach is really adequate for building production programs.

Prolog-II is missing some "inessential" features of earlier PROLOGs. Integers cannot be negative, though real numbers can be. Several layers of syntactic sugar have been removed. The Prolog-II list syntax resembles the basic dot syntax that earlier PROLOGs and LISP hide with a more readable list notation. The omission of operators makes many programs that manipulate symbolic structures less readable. The variable naming rules and the restriction on placement of comments interact to require a coding style in which initial block comments and mnemonic rule names are the main aids to creating readable programs. The sample programs in *PROLOG* and the Prolog-II manual use sparse one-line header comments and one- or two-character variable names.

Prolog-II is a different language from the other PROLOGs. The syntax differs from Edinburgh and Standard in almost every respect: the terminology is different, and many of the language features are different, new, or missing. Converting an Edinburgh PROLOG program to Prolog-II is on the same order of difficulty as converting a C program to Modula-2. Prolog-II is a new language with powerful new features. It is theoretically more powerful than older PROLOGs,

	PROLOG/m	AAIS Prolog	MacPROLOG	Prolog-II
Atoms	yes	yes	yes	yes
Clauses	yes	yes	yes	yes
Rules	yes	yes	yes	yes
Lists	yes	yes	yes	yes
Characters	no	yes	no	yes
Integers	yes	yes	yes	yes[1]
# of bits	16	32	28[2]	24
Floats	yes	yes	yes	yes
# of bits	64	64	64	64[3]
Strings	no	yes	no	yes
Arrays	no	no	no	yes
I/O streams	no	yes	no	no
Buffers	no	no	no	yes
Infinite trees	no	no	no	yes

1. Nonnegative integers only.
2. Documentation states range is −99,999,999 to +99,999,999.
3. Inferred from external procedure conventions.

Table 2: *Data types*

	PROLOG/m	AAIS Prolog	MacPROLOG	Prolog-II
Multistream	no	yes	yes	no
Random	no	yes	yes	no
Backtrackable input	no	yes	no	no
Pretty printing	no	yes	no	no
Tree drawing	no	no	no	yes

Table 3: *Input/output extensions*

	PROLOG/m	AAIS Prolog	MacPROLOG Standard	MacPROLOG Edinburgh	Prolog-II
Cut	!	!	/	!	/
Backtracking	yes	yes	yes	yes	yes
Negation by failure	not	not	NOT	not	—[1]
Disjunction	;	;	OR	;	—
If	->	->	IF	->	default
Else	; or |	;	[2]	;	[2]
Iteration over	repeat	repeat	—	repeat	—
Solutions	forall[3]	—	FORALL	—	—
Lists	—	foreach foreachlist	MAP	map	—
Numbers	—	for	—	—	—
Collection	bagof[3] setof[3]	bagof setof	BAGOF SETOF ISALL	bagof setof findall	— list-of
Delayed evaluation	— —	— —	— TOHOLLOW TOGROUND	—	freeze dif
Inequality	—	—	—	—	dif
Signals	—	—	[4]	[4]	block block-exit
Dynamic invocation	call	call apply		call	—
Metavariables	X	X	_X[5]	X	—

1. Example in the primer—not normally defined.
2. Part of the *If* predicate.
3. In the TOOLBOX.
4. Program-defined error handlers for each error number.
5. The MacPROLOG metavariable facility is exceptionally powerful.

Table 4: *Control features*

MAC PROLOGS
(continued from page 37)

though I feel it is less readable.

Features

Tables 2–6, pages 37 and 38, summarize the language features of the different PROLOGs. A few entries need more explanation.

Backtracking

Backtrackable input (Table 3) involves one of the traditional problems in PROLOG design—how to reconcile the side-effect-free, backtracking world of logic programming with the side-effect- and state-based external world. The PROLOG language compromises by making I/O operations nonbacktrackable. When *foo(S)* fails in:

try(S) :– read(S), foo(S).

PROLOG will not backtrack and retry *read(S)*. Although this strategy is usually fine, it does make some loops harder to write (what if you wanted to keep reading clauses until you

	PROLOG/m	AAIS Prolog	MacPROLOG	Prolog-II
Database				
modification	yes	yes	yes[1]	yes
Rule access	yes	yes	yes	yes
Keyed database	no	yes[2]	no	no
Assignment	no	set_global	remember	assign
Read variable	no	get_global	recall, default	val
Deletion	no	no	forget	no
Properties	no	no	yes	no

1. To a separate interpreted database.
2. Imperfect emulation of the Prolog-10 feature because of the different (hashed) internal structure of the AAIS database.

Table 5: *Side-effect features*

	PROLOG/m	AAIS Prolog	MacPROLOG	Prolog-II
Modules	no	packages	no[1]	worlds
Grammar	no	yes	no	no
Full char set	no	no	yes[2]	partial[3]
Mac interface	no	yes[4]	yes	a little
Graphics	no	yes[4]	no	yes
Other toolbox	no	yes[4]	no	no
Other language	no	yes[5]	no	yes[6]
Click icon	yes	yes	yes	yes
Saves system	no	no	yes	yes
Run-time system	no	no	yes[7]	no

1. The primer mentions a module system in other versions of micro-Prolog.
2. No choice of font; extended chars are not alphanumerics.
3. Special font (enhanced Monaco) provided.
4. Direct, complete support for the Mac Toolbox.
5. Like Toolbox support, but for arbitrary code resources.
6. Requires Lisa Workshop.
7. Special license required for redistribution.

Table 6: *Other features*

	PROLOG/m	AAIS Prolog	MacPROLOG interp	MacPROLOG comp	MacPROLOG opt	Prolog-II
Reverse 30	11.8	0.7	2:13.1*	2.1	1.4	1.8
LIPS	42	709	—	236	354	276
Reverse failed	50	60	30	90	240	50
Map coloring	4:12.0	14.3	4:05.6	8.0	6.2	40.0
With dif						7.7
Graph connect	4.2	0.7	4.8	1.4	1.3	
Sieve 100	26.2	3.3	2:48.4*	10.7	10.7	5.7

* Ran out of memory after listed time.

Table 7: *Benchmark results (minutes: seconds)*

found one that satisfied *foo(S)*?). AAIS Prolog provides an additional set of input predicates that retry when backtracked to.

Control Structures

Table 4 has a lot of information crammed into it. Each row represents a control structure. The entries in each row are the name(s) of the control structure in that PROLOG, — means the control structure isn't supported in that product, and an empty space means that this row is a continuation of the preceding one because some product has more than one distinct predicate for that control structure. Negation by failure is the weak analogue of true negation used in PROLOG. Dynamic invocation is the ability to execute a clause bound to a variable. Metavariables are an extension of dynamic invocation—most PROLOGs treat a variable found where a clause would be expected as implicit dynamic invocation. MacPROLOG also allows metavariables in other positions, such as a metavariable as all or part of the body of a clause.

Side Effects

PROLOG is mainly a side-effect-free language; good PROLOG programs make minimal use of side effects. But many programs can't be written without some side effects (see Table 5). The first two items refer to the earliest form of PROLOG side effects. PROLOG rules and facts are stored together in a memory database. The earliest PROLOGs provided special predicates to access and modify this database in order to bootstrap up the PROLOG environment. Of course, programmers began to use these predicates for other reasons such as including self-modifing code. Although excessive use of the database predicates is slow,

Books

Bratko, Ivan. *Prolog Programming for Artificial Intelligence.* Reading, Mass.: Addison-Wesley, 1986. Paperback, 423 pages.

This is a new book that will be included as part of AAIS Prolog by the time you read this review. My copy hasn't arrived yet, so I can't say anything more about it.

Clark, K. L., and McCabe, F. G. *micro-PROLOG.* Englewood Cliffs, N.J.: Prentice-Hall, 1984. Paperback, 401 pages.

Although there are several micro-PROLOG primers, this is the official one. It is also the only one I know of that goes beyond elementary use of the Simple syntax to teach the possibilities of the whole language. The last section of the book, Applications of micro-PROLOG, consists of individual articles by different authors, including a critical-path-analysis program; two chapters on search, pruning, and game playing; and, you guessed it, the obligatory expert system example.

Clocksin, W. F., and Mellish, C. S. *Programming in Prolog.* 1st ed. Berlin: Springer-Verlag, 1981. Paperback, 279 pages.

This is the classic PROLOG text. The first PROLOG textbook, it is one of the main reasons for the "Edinburgh standard." This standard is really the core PROLOG that Clocksin and Mellish derived from the existing Edinburgh PROLOG systems for this text. Although of great historic importance, later and better PROLOG texts are now available.

Giannesini, Francis; Kanoui, Henry; Pasero, Robert; and van Caneghem, Michal. *PROLOG.* Reading, Mass.: Addison-Wesley, 1986. Paperback, 260 pages.

This is the Prolog-II primer. It is clearly written and quite readable. The last chapter includes a good deal of material on parsing natural languages and compiling grammars as well as the obligatory expert system example. If you're interested in Prolog-II, this is the best introduction you'll likely to get. If you buy Exper-Prolog-II, you get it as part of the package.

Sterling, Leon, and Shapiro, Ehud. *The Art of Prolog.* Cambridge, Mass.: MIT Press, 1986. Hardback, 427 pages.

This is the book I am using to learn PROLOG. It is an excellent first text for experienced programmers who don't want to bother with a primer. It is also an excellent classroom text. The book is divided into four roughly equal parts: Logic Programming, The Prolog Language, Advanced Prolog Programming Techniques, and Applications.

The first part, Logic Programming, is one of the book's greatest strengths and weaknesses at the same time. Its goal is to teach the ideas and techniques of logic programming before getting in to the quirky details and hackery of a specific language. The problem is that it is full of logic programs that look exactly like the PROLOG programs in the book. Many of the logic programs such as:

```
plus(0,X,X) <-
    natural_number(X).
plus(s(X),Y,s(Z)) <-
    plus(X,Y,Z).
natural_number(0).
natural_number(s(X)) <-
    natural_number(X).
```

cannot execute in PROLOG. Novices studying alone are likely to enter these programs and become very confused when they don't work.

Advanced Prolog Programming Techniques develops examples such as Eliza, a parser for grammar rules, an alpha-beta pruning search, a PROLOG tracer, and the obligatory simple expert system shell.

The Applications section of the book consists of the chapters "Game Playing Programs" (Mastermind, Nim, and Kalah), "A Credit Evaluation Expert System," "An Equation Solver," and "A Compiler" (to an abstract assembly language).

The second main flaw in this book is the number of typos, especially in the programs. I suspect that this may be partly a result of the new process used to produce this book. The authors and assistants produced the book using TEX and sent the camera-ready copy to MIT Press, which published it without further ado. I'm sure this new process is faster and cheaper, and it certainly produced an attractive book, but I wonder if the increased speed has dropped too many proofreading steps.

All in all, I highly recommend this book to anyone who is familiar enough with programming in general to avoid the pitfalls and to PROLOG beginners looking for a more advanced book to follow their primer.

Dr. Dobb's Journal of Software Tools . . . the R_x for programmers

SUBSCRIBE AND SAVE

Subscribe to **Dr. Dobb's Journal** and save over $5—a 15% savings off the cover price!

__Please charge my: __ Visa __ MC __ American Express

__ Payment enclosed __ Bill me later

Card # __________ Exp. date __________

Signature __________

Name __________

Address __________

City __________ State _____ Zip _____

ONLY $29.97

Canada & Mexico add $10 for surface mail. All other countries add $27 for airmail. All foreign subscriptions must be pre-paid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc.

344

SUBSCRIBE AND SAVE

Subscribe to **Dr. Dobb's Journal** and save over $5—a 15% savings off the cover price!

__Please charge my: __ Visa __ MC __ American Express

__ Payment enclosed __ Bill me later

Card # __________ Exp. date __________

Signature __________

Name __________

Address __________

City __________ State _____ Zip _____

ONLY $29.97

Canada & Mexico add $10 for surface mail. All other countries add $27 for airmail. All foreign subscriptions must be pre-paid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc.

344

Comments & Suggestions

Dear Reader, **April 1987, #126**

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take the time to write a letter. This card provides you with a quick and easy way to correspond and, if you include your name and address, we may use appropriate comments in The Letters column. Simply fill it out and drop it in the mail. —Ed.

Which articles or departments did you enjoy the most this month? Why?

____

____

____

Comments or suggestions: __________

____

____

____

Name: __________

Address: __________

____

34

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

P.O. BOX 27809
SAN DIEGO, CA 92128

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

P.O. BOX 27809
SAN DIEGO, CA 92128

PLACE STAMP HERE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

501 GALVESTON DR.
REDWOOD CITY, CA 94063

Dr. Dobb's Journal of Software Tools . . .
the Rx for programmers

MAC PROLOGS
(continued from page 39)

hard to understand, and generally bad form, some use of them has true advantages in both speed and power. Such powerful predicates as *setof* would not be possible without database modification.

Other Features

Grammar (Table 6) refers to definitive clause grammar rules. This special syntax for defining a useful subset of natural-language grammars is described in *Programming in Prolog* and *The Art of Prolog*.

None of the PROLOGs can create true stand-alone applications. Click icon means that a program icon can be double-clicked to automatically start the PROLOG with that program loaded. All the PROLOGs permit the loaded program to take control as it starts for a poor-man's application.

Vendors

PROLOG/m
Chalcedony Software Inc.
5580 La Jolla Blvd., Ste. 126
La Jolla, CA 92307
(619) 483-8517
$99.95, PROLOG/m
$29.95, TOOLBOX
$29.95, TOYBOX
$49.95, NFL-X-pert
Reader Service Number 27

AAIS Prolog
Advanced A.I. Systems Inc.
P.O. Box 39-0360
Mountain View, CA 94039
(415) 961-1121
$150
Reader Service Number 28

MacPROLOG
Programming Logic Systems Inc.
31 Crescent Dr.
Milford, CT 06460
(203) 877-7988
$295, MacPROLOG
$100, optimizing compiler
Reader Service Number 29

ExperProlog-II
ExperTelligence Inc.
559 San Ysidro Rd.
Santa Barbara, CA 93108
(805) 969-7871
$495
Reader Service Number 30

Prolog-II allows the entire state of the PROLOG to be saved in binary form so that a development or application can be restarted quickly. MacPROLOG provides two minimal run-time systems, but the run-time system, MacPROLOG itself, and your compiled code must all be on the disk in order to run the program.

The Benchmarks

The most common metric of PROLOG performance is LIPS, logical inferences per second. This is no more useful than MIPS, but I've done it anyway. The second row of Table 7, page 38, is a LIPS figure calculated by dividing the time for naive reverse of 30 elements—the most common PROLOG benchmark. The third row of Table 7 is a measure of memory utilization. It is the length of a list that caused a memory or stack-full error message from naive reverse. List sizes of 30, 50, 60, 90, 120, 180, and 240 were used in the test.

Map coloring is a simple generate-and-test recursive loop. This sort of thing is fairly common in PROLOG applications. Graph connect uses a lot of database hacking with *asserta* and *retract*. Sieve is, of course, the traditional *Byte* sieve benchmark. I included it to give some idea of the arithmetic performance of the PROLOGs.

AAIS Prolog comes up with about twice the LIPS of the MacPROLOG optimizing compiler here, but the ratings are reversed with a more complicated benchmark. Both the MacPROLOG compilers handily won the map coloring test. The compilers, especially the optimizing compiler, had much better memory utilization than any of the interpreters. The number of resolutions in a naive reverse of length N is the triangular number of $N+1$—that is $(N+1)(ceiling((N+1)/2))$. Reverse of 30 takes 496 resolutions, 60 takes 1,891, 90 takes 4,186, and 240 takes 29,161 resolutions. The advantage of *dif* is also clear, improving Prolog-II's performance on map coloring by a factor of 5.

Conclusions

PROLOG takes a lot of memory. All these products are really cramped on a 512K Macintosh.

These are four very different products. For learning PROLOG and playing with it at home, I'd recommend AAIS Prolog. It's fast; has many features, including unlimited potential use of the Mac environment; and the price is reasonable. For serious research, development, or prototyping in which graphics are not an issue, I'd definitely consider MacPROLOG as well. Its memory-efficient compilers and easy use of the Mac user interface make it well worth considering. Prolog-II has a powerful and interesting base language, but the price is high and both the user interface and language are less convenient than the other PROLOGs. Buy it if you like the Prolog-II language model or need its features; otherwise, wait until its vendor lowers the price. The only PROLOG I cannot recommend at all is PROLOG/m. It's not bad by itself, but it's just not competitive. For only a little more money you can get AAIS Prolog.

If you are determined to ship a Macintosh product in PROLOG, you need two features: the ability to supply your own user interface and some sort of run-time system. Only MacPROLOG comes close to meeting these requirements. It meets the first requirement easily, but I'm not convinced by a "run-time system" that requires that both the language interpreter and the run-time system file go along with the application on every disk.

Notes

1. J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *Journal of the ACM* 12 (January 1965): 23–41.
2. C. E. Hewitt, "PLANNER: A Language for Proving Theorems in Robots," *Proceedings of IJCAI-69* (Washington, D.C.: International Joint Conference on Artificial Intelligence, May 1969): 10.
3. C. E. Hewitt, "Concurrency in Intelligent Systems," *AI Expert* (San Francisco: CL Publications, 1986): 44–50.
4. E. Shapiro, *Concurrent Prolog* (Cambridge, Mass.: MIT Press, 1987).
5. K. Kahn, E. D. Tribble, M. Miller, and D. Bobrow, "Objects in Concurrent Logic Programming Languages," *OOPSLA'86 Conference Proceedings*: 242–257.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 3.**

MYCIN-Like Expert Systems

by Richard W. Grigonis

An approach to setting up an expert system with BASIC

For Amiga owners interested in artificial intelligence but lacking Amiga LISP, Metacomco's Cambridge LISP 68000, XLISP, or Micro Forge PROLOG, here is an interesting approach to setting up a quick expert system using the Amiga BASIC package that you already own. By now you've probably noticed that this is the best Microsoft BASIC interpreter ever written (so good, in fact, that it almost isn't BASIC), and further improvements are on the way.

Rule-Based Systems

As all readers of *DDJ*'s special AI issues are aware, most expert systems are essentially a collection of production rules and are therefore known as rule-based systems (RBSs). The rules have a left-hand side (the antecedent, or propositions combined in logical AND/OR form that comprise a situation) and a right-hand side (the consequent, goal, or action to be taken).

A simple rule is as follows:

```
IF Left-hand side
  THEN Right-hand side
IF X has hair
  THEN X is a mammal.
```

Notice how the rules of an RBS are simple modules of logic, taking the form of *IF...THEN* statements. The *THEN* sections of some rules match the *IF* sections of others, forming an inference network that can be drawn as an AND/OR tree. The rules themselves are quite useless unless the program also possesses an inference engine that searches through the rules in one of two ways. If the situation on the left side is examined first and the right side is taken as the action, then the system can be said to be a bottom-up, forward-chaining one. If, however, the right side is examined first and taken as a goal or hypothesis to be proved by demonstrating the truth of the propositions on the left side, then such a system is a top-down or backward-chaining one.

MYCIN for the Masses

Perhaps the most famous backward chaining, depth-first search expert system is MYCIN, which was also one of the first major expert systems and is still used as a sort of benchmark in comparing expert systems. MYCIN has 450 rules that are used to diagnose and suggest antibiotic treatment for 100 blood and meningitis infections. The program listing accompanying this article (Listing One, page 74) is a simple MYCIN-like program written as a demonstration of such a system in Amiga BASIC. It took just a few hours to port the original version to the Amiga from another one of Jay Miner's hardware creations, my old Atari 800. In fact, a few line numbers from the original still exist in the Amiga BASIC code, serving as labels.

Instead of diagnosing diseases on the basis of symptoms, the program in the listing identifies animals on the basis of physical attributes and observed behaviors. It is a toy system designed to demonstrate the MYCIN reasoning mechanism in Amiga BASIC. You can find the knowledge base of rules it uses on pages 242–243 of Winston and Horn's textbook *LISP*.[1]

The problem with writing expert systems in BASIC as opposed to PROLOG is that the inference engine necessary to parse the knowledge base of rules and to drive a path through them must be written by the programmer.

You can store production rules as *DATA* statements as follows:

```
DATA Rule 1,IF,has hair,
  THEN,is mammal
DATA Rule 2,IF,gives milk,
  THEN,is mammal
```

Unfortunately, as most BASICs are not recursive, programming a general-purpose top-down parser to act upon these *DATA* statements requires considerable effort. You must establish push-down stacks, queues, stack conventions, and so on to store local variables for the reentrant subroutines of the inference mechanism. Another problem with a general-purpose parser of this type is that the processing overhead needed to search through the knowledge base of production rules and keep track of what is going on results in a slow program, although it is versatile. Code for a BASIC expert system using *DATA* statements such as the preceding ones, along with a stack, backward-chaining, and the Winston and Horn rules (but without the MYCIN certainty factor inference routines) can be found in the September 1981 issue of *Byte*.[2]

Elegant BASIC?

The (supposedly) quick-and-dirty approach I have elected to use for developing an Amiga BASIC expert system requires a bit more code for each rule, but it forces the BASIC interpret-

Richard W. Grigonis, 49 Haring St., Bergenfield, NJ 07621. Richard is a freelance programming consultant.

er (via its own stack) to handle the bookkeeping required for searching through the rules in a top-down, backward-chaining, depth-first manner. A single stack is employed, but only to keep track of what routines have been activated so that the system can explain its reasoning to users when they key in "why" or "why?" in response to a question from the system. Otherwise, the expert system is herein presented in a form similar to—though not exactly the same as—a syntax-directed recognizer, such as those used in computer language interpreters and compilers.

A BASIC interpreter, for example, can be constructed from these production rules:

<statement> ::=
LET <identifier> = <expression>
<statement> ::=
NEXT <identifier>
<statement> ::=
INPUT <identifier list>
<statement> ::=
GOTO <line number>
<statement> ::=
FOR <identifier> = <expression>
TO <expression>
<statement> ::=
GOSUB <line number>

Systems programmers writing a BASIC interpreter based upon these rules must now write a syntax-directed recognizer in C or assembly language that can parse all the kinds of BASIC language statements shown above. They do this by writing separate recognizer procedures for each nonterminal in the language, with one procedure calling others as dictated by the production rules. For the preceding rules pertaining to BASIC statements, John Zarrella[3] suggests one possible syntax recognition procedure (see Example 1, right).

Because an expert system is also based upon production rules, programmers can examine the components of each rule and write recognizer procedures or subroutines in a high-level language in the same manner as with the BASIC interpreter discussed previously.

In constructing such a program, you must write a subroutine for each nonterminal and terminal in the language or, as in this case, each hypothetical fact, assertion, or combined assertion to be proven. The "words" analyzed by this expert system syntax-directed recognizer are the numeric values supplied by users in response to questions asked by the program. In other words, each hypothesis (albatross, penguin, ostrich, zebra, and so on) is a subroutine "proved" by calling other subhypotheses (bird, mammal, and so on), also in the form of subroutines, that in turn call still other hypothesis/subroutines (has feathers, lays eggs, and so on). If you could call such procedures recursively (which is not required in this kind of system), then you would have a recursive descent parser—not exactly what you would call quick-and-dirty code.

Such syntax-directed recognizers are not general purpose—meaning that you cannot simply plug in a new set of rules describing expertise in some other domain of knowledge—but they are faster than general-purpose inference engines and they are more in keeping with the procedural and modular knowledge representation philosophy suggested by production rule systems. Indeed, you can now insert additional, special subroutines or functions of arbitrary complexity here and there in the program as needed. Such systems allow the BASIC language to do what it does best—define the heuristic flow of control (procedures) of the expert system. It does this so well that you can no longer refer to the production rules explicitly because they are now implied in the pattern of nested subroutines residing in the code. You must therefore now speak not just of rules and combined assertions but of "facts" and "combined facts" in numeric arrays worked upon by the subroutines, all of which can be imagined as residing on the nodes and arcs of an imaginary inference net or AND/OR tree.

Another strength of this kind of system is that error messages and explanations of the top-down reasoning process are easier to program as the system is designed specifically for the particular knowledge base used.

Like MYCIN, the system presented here can accept information volunteered by users at any level of the reasoning process. If users are not absolutely sure of the animal or classification they are thinking of, the program ignores their input and digs deeper into the AND/OR tree. Provision has been made in the program for programmers to change this threshold easily (see the *IF* statement two lines above the *Test.for.a.positive.number:* subroutine).

AND Clauses

The system can handle negative inferences and degrees of certainty in a user's answers through a mathematical process essentially the same as that used by MYCIN.

```
procedure STATEMENT;
   local LEXEME;
   LEXEME=GETLEXEME;
   select LEXEME of
"LET":        begin
              call IDENTIFIER;
              IF GETLEXEME /=   "=" then call ERROR;
              call EXPRESSION;
              end;
"NEXT":       call IDENTIFIER
"INPUT":      call IDENTIFIERLIST;
"GOTO":       call LINENUMBER;
"FOR":        begin
              call IDENTIFIER;
              if GETLEXEME /=   "=" then call ERROR;
              call EXPRESSION;
              if GETLEXEME /=   "TO" then call ERROR;
              call EXPRESSION;
              end;
"GOSUB":      Call LINENUMBER;
end;
```

Example 1: *John Zarella's syntax recognizer*

(continued from page 43)

As an example, let's take a look at one of the rules in the system's AND/OR tree:

IF animal is an UNGULATE
AND animal HAS BLACK STRIPES
THEN animal is a ZEBRA
(attenuation factor=0.8)

Let's say that the user's certainty on the *UNGULATE* branch of the *AND*ed relation is 0.7 (1.0 being absolute certainty) and the certainty on the *HAS BLACK STRIPES* branch is 0.8.

In normal probability theory, you would multiply the individual fractional probabilities, yielding 0.56. MYCIN, however, does not use standard probability theory. Conventional probability theory was rejected because it was felt that the *AND* clauses in classification systems violate the two foundations of standard probability theory (particularly Bayes' rule): statistical independence and prior probabilities (or priors).

Standard statistical probability assumes that the components in the *AND*ed relations are independent of each other and that an examination of a sufficiently large number of examples of a rule allows you to construct a statistical, frequency model of the rule so that you can give an a priori probability of a hypothesis being true in the absence of any evidence. The developers of MYCIN rejected and/or modified these ideas because the symptoms of a disease (or the physical attributes of an animal, for that matter) are not independent but usually occur in groups. Also, it is difficult to obtain data on and analyze thousands of cases to determine frequencies.

Instead, MYCIN's developers created their own technique for dealing with uncertainty, based on confirmation theory (logical probability) and the use of certainty factors and attenuation factors. You must therefore distinguish certainty (the degree of confidence you have in a fact or rule) from ordinary probability.

A certainty factor (CF) is a number between −1 and 1 given to a fact or relation to indicate the confidence a user has in providing data concerning a fact or relation to the expert system. In this sense, certainty factors are really confidence factors, not probability coefficients. By the end of a user's session with an expert system, the program itself has combined and computed new certainty factors. In MYCIN, if the computed truth of a fact exceeds 0.8, then the fact is judged to be proven and the certainty factor is now 1.0. Also, if the certainty factor falls into the range −0.2 to 0.2, then the certainty factor is set to 0 (unknown) and a certainty factor in the range −0.8 to −1.0 is converted to −1.0 (definitely false).

The program combines and computes new certainty factors.

An attenuation factor is a number between 0 and 1 that is multiplied by a certainty factor, yielding a new certainty factor. It is an indicator of a rule's inherent reliability, or at least the confidence a human expert had in the efficacy of the rule when the system was being developed. Just as a certainty factor starts out as really a confidence factor on the part of the user, an attenuation factor is likewise a confidence factor on the part of the human expert from whom the rules were derived.

In the *AND* clause shown earlier, the probability of one conditional *AND* another is taken as a minimum of their certainties, so the program finds the lowest certainty factor on the branches of the *AND* clause (the certainty factors of *UNGULATE* or *HAS BLACK STRIPES*) and multiplies it by the attenuation factor of 0.8. If the certainty passed up the tree by the *Prove.ungulate:* subroutine is 0.7 and the certainty factor given by the *Prove.black.stripes:* subroutine is 0.8, the *Prove.zebra:* subroutine will select the lower figure of 0.7 and multiply it by the zebra *AND* clause attenuation of 0.8, yielding a new output certainty factor of 0.56. The figure, coincidentally, matches that given by standard probability. In any event, this result is passed up to the user as the final certainty factor for the animal being a zebra.

OR Clauses

But what would have been the case if the rule had been an *OR* clause instead of an *AND* clause? That is, what if the rule had looked like this:

IF animal is an UNGULATE
OR animal HAS BLACK STRIPES
THEN animal is a ZEBRA

The designers of MYCIN thought that the certainty factors on the branches of an *OR* node should reinforce one another. Remember that the certainty factor on the *UNGULATE* branch is currently 0.7 and the certainty factor on the *HAS BLACK STRIPES* branch is 0.8. The CF on the *UNGULATE* branch takes you 70 percent toward proving that the animal is a zebra (0.7). That still leaves 30 percent (0.3) to go in order to achieve absolute certainty. As it happens, the CF on the *HAS BLACK STRIPES* branch (0.8) "carries" the the total certainty an additional 80 percent over the remaining distance of 0.3. Because 80 percent of 0.3 is 0.24, the certainty factor of the animal being a zebra is now 0.7 + 0.24, or 0.94. Had a third branch existed with a certainty factor of, say, 0.5, then the total certainty would have been carried another 50 percent over the remaining distance of 0.06, yielding a new total certainty factor of 0.97.

One way of expressing this procedure mathematically is as follows:

New CF = CF1 + CF2(1 − CF1)

A more confusing (though equivalent) expression is this one:

New CF = CF1 + CF2 − (CF1 × CF2)

In order to handle negative numbers, however, you would have to convert the above equation into the following:

New CF = CF1 + CF2 + (CF1 × CF2)

Also, things get awkward when you are using three certainty factors:

New CF = CF1 + CF2(1 − CF1)

EXPERT SYSTEMS
(continued from page 44)

$$+ CF3(1 - [CF1 + CF2(1 - CF1)])$$

Fortunately, this equation can be simplified to this one:

$$\text{New CF} = 1 - (1 - CF1)(1 - CF2)(1 - CF3)$$

My system uses this equation if at least one certainty factor on a branch is positive. If all the certainty factors are negative or 0, the system uses this equation:

$$\text{New CF} = -1 + (1 + CF1)(1 + CF2)(1 + CF3)$$

At this point you should distinguish the use of negative numbers in stating the confidence in a hypothesis from situations where a lack of evidence is necessary to prove a hypothesis, in which case you must add a rule to the knowledge base testing for the lack of certain information. Here is one such rule from MYCIN itself:

```
IF identity of organism
   is not known
AND gram stain of organism
   is not known
AND morphology of organism
   is not known
AND site of culture is csf
AND infection is meningitis
AND age of patient
   is less than or equal to 17
THEN (.3) category of organism
   is enterobacteriaceae.
```

Some researchers have pointed out some deficiencies with the MYCIN approach to reasoning with certainty factors.[4] I find that all forms of the *OR* clause equations increase the certainty factor values too much, so I place attenuations on all possible branches to bring the results closer to what you would expect from standard probability theory. Researchers now use formulas closer to standard probability theory in the new systems, but I will not examine them here.

Expanding the System

The system is currently designed to identify seven animals, and the *HYPOTHESIS* numeric array has been dimensioned to accept up to 20 animals. I'll now demonstrate how you add a new animal to the system.

Let's say you want the program to be able to recognize an emu, which is a large flightless bird like an ostrich but with dark feathers and large red eyes. The rule you wish to express is as follows:

```
IF the animal is a BIRD
AND the animal CANNOT FLY
AND the animal HAS DARK FEATHERS
AND the animal HAS BIG RED EYES
THEN the animal is an EMU
```

This *AND* clause looks as if its going to be pretty reliable, so let's give it an attenuation of 1.

Remember that, as the program is backward chaining, it looks at this rule in reverse, taking the emu identity as a hypothesis to be proved by calling other subroutines that in turn attempt to prove that the animal is a bird, cannot fly, and so on.

Let's add the actual emu subroutine first. First, you scroll down to the bottom of the *DATA* statements, where you see the following lines:

```
DATA 35,has pointed teeth
```

pointed.teeth=35
DATA −1, END OF DATA

Now add the new fact and user request string for *emu* (fact #36) above the last line. The result should look like this:

DATA 35,has pointed teeth
pointed.teeth=35
DATA 36,is an emu
emu=36
DATA −1, END OF DATA

As *emu* is one of the top-level hypotheses (animals to be identified), there are two special areas in the program to change. Below the line, *REM TOP-LEVEL HYPOTHESES (ROOTS) OF AND/OR TREE:*, find these lines:

HYPOTHESIS(7)=cheetah
number.of.hypotheses=7

and change them to look like these:

HYPOTHESIS(7)=cheetah
HYPOTHESIS(8)=emu
number.of.hypotheses=8

The other area of the program to change as a result of *emu* being a top-level hypothesis is the executive calling routine a little farther down. Make an addition just above line 10165 that reads as follows:

GOSUB Prove.emu
IF halt.on.success=2
  AND OUTPUT.CF(emu)=1 THEN 10165

You then add the subroutine shown in Example 2, below, to prove the animal is an emu (with an attenuation of 1) to the very bottom of the program.

If you had been trying to represent an *OR* clause instead of an *AND* clause:

IF the animal is a BIRD
OR the animal CANNOT FLY
OR the animal HAS DARK FEATHERS
OR the animal HAS BIG RED EYES
THEN the animal is an EMU

you would have had to assign attenuations to each branch because the components of such a disjunctive relation can each be considered as a separate minirule contributing a certainty factor to prove that the animal is an emu. This is known as a multiply argued certainty. A subroutine describing this (again with arbitrarily chosen attenuations) would look like that in Example 3, page 48.

But what if the rule had taken the form of a compound relationship of *AND*s and *OR*s?

IF (animal is a BIRD
  AND animal CANNOT FLY
  AND animal HAS DARK FEATHERS)
OR animal HAS BIG RED EYES
THEN animal is an EMU

This way of proving the animal is an emu is actually just an *OR* clause with two components, one of which can be further reduced to some *AND*s. My solution is to write a separate routine for the *AND* clause (with an additional attenuation to keep the *OR* clause certainty under control). Thus, you now have the two subroutines

```
Prove.emu:
  current.fact=emu
  GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.bird
  GOSUB Prove.cannot.fly
  GOSUB Prove.dark.feathers
  GOSUB Prove.big.red.eyes
    number.of.and.clause.components=4
      AND.COMPONENT(1)=bird
      AND.COMPONENT(2)=cannot.fly
      AND.COMPONENT(3)=dark.feathers
      AND.COMPONENT(4)=big.red.eyes
    at.factor.for.AND.clause=1
    GOSUB Compute.AND.clause.cf
  GOSUB Deduce
RETURN
```

Example 2: *Emu-proving subroutine*

(continued from page 47)

shown in Example 4, below.

Supposedly, the Amiga BASIC interpreter does not accept subroutine labels longer than 40 characters, but the 43-character label *Prove.bird.-and.cannot.fly.and.dark.feathers:* works correctly. The Microsoft people probably mean that the first 40 characters of the label are significant.

The second subroutine will also require access to both the user and the numeric arrays to compute the certainty factors, so you must log it in your *DATA* statements:

```
DATA 37,is a bird and cannot fly
   and dark feathers
bird.and.cannot.fly
   .and.dark.feathers=37
```

```
Prove.emu:
  current.fact=emu
  GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.bird
  GOSUB Prove.cannot.fly
  GOSUB Prove.dark.feathers
  GOSUB Prove.big.red.eyes
    number.of.or.clause.components=4
      OR.COMPONENT(1)=bird
        AT.FACTOR.FOR.OR.COMPONENT(1)=.8
      OR.COMPONENT(2)=cannot.fly
        AT.FACTOR.FOR.OR.COMPONENT(2)=.85
      OR.COMPONENT(3)=dark.feathers
        AT.FACTOR.FOR.OR.COMPONENT(3)=.9
      FOR.COMPONENT(4)=big.red.eyes
        AT.FACTOR.FOR.OR.COMPONENT(4)=1
    GOSUB Compute.or.clause.cf
  GOSUB Deduce
RETURN
```

Example 3: *Alternate emu-proving subroutine*

```
Prove.emu:
  current.fact=emu
  GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.bird.and.cannot.fly.and.dark.feathers
  GOSUB Prove.big.red.eyes
    number.of.or.clause.components=2
      OR.COMPONENT(1)=bird.and.cannot.fly.and.dark.feathers
        AT.FACTOR.FOR.OR.COMPONENT(1)=.8
      OR.COMPONENT(2)=big.red.eyes
        AT.FACTOR.FOR.OR.COMPONENT(2)=.85
    GOSUB Compute.or.clause.cf
  GOSUB Deduce
RETURN

Prove.bird.and.cannot.fly.and.dark.feathers:
  current.fact=bird.and.cannot.fly.and.dark.feathers
  GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.bird
  GOSUB Prove.cannot.fly
  GOSUB Prove.dark.feathers
    number.of.and.clause.components=3
      AND.COMPONENT(1)=bird
      AND.COMPONENT(2)=cannot.fly
      AND.COMPONENT(3)=dark.feathers
    at.factor.for.and.clause=1
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN
```

Example 4: *Emu-proving routines for compound case*

So much for the emu subroutine.

As for the other subroutines that *Prove.emu:* calls, you already have those that attempt to prove that the animal is a bird and cannot fly, but you need two subroutines to prove that the animal has dark feathers and big red eyes. These are both terminals (they don't have to call other parsing subroutines—they just ask the user a question), so they are easier to write. First, you add this new set of *DATA* statements near the top:

```
DATA 38,has dark feathers
dark.feathers=38

DATA 39,has big red eyes
big.red.eyes=39
```

Having done this, you scroll to the bottom of the program and add two subroutines:

```
Prove.dark.feathers:
   current.fact=dark.feathers
   GOSUB Test.fact.for.human.input
   IF leave=yes THEN RETURN
   GOSUB Deduce
RETURN

Prove.big.red.eyes:
   current.fact=big.red.eyes
   GOSUB Test.fact.for.human.input
   IF leave=yes THEN RETURN
   GOSUB Deduce
RETURN
```

That's it.

By writing some templates of various sizes for the subroutines handling *AND*s and *OR*s, you can copy them as needed and quickly assemble a MYCIN-like expert system running under the Amiga BASIC (Microsoft BASIC) interpreter. The system currently can handle rules with clauses having 12 *AND*s, 12 *OR*s, or a combination of both. You can change this easily by redimensioning the *AND.COMPONENT*, *OR.COMPONENT*, and *AT.FACTOR.FOR.OR.COMPONENT* arrays.

You might experiment by inserting additional subroutines that perform other mathematical tests outside the MYCIN environment (go ahead, it's OK—most mathematicians think MYCIN's mathematical reasoning is pretty ad hoc anyway, even though it has been known to outperform experts, which isn't saying much for experts!). You might also consider adding subroutines allowing the program to explain how a certainty factor was reached after the conclusion of each fact by listing the certainty factors passed up from those below it in the AND/OR hierarchy.

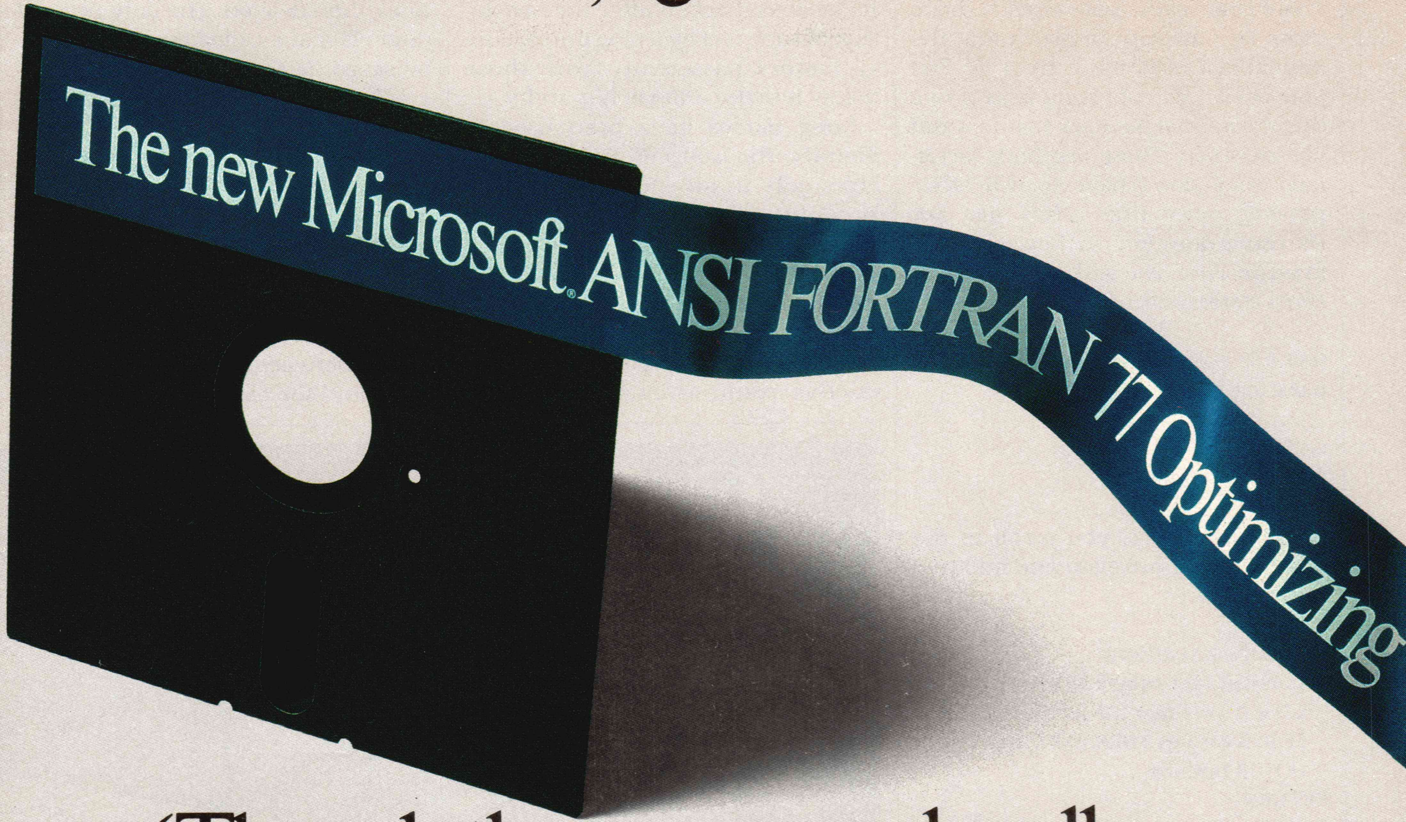
Long names have been used to identify the subroutines and variables only in an effort to make the program's flow of control understandable. By shortening these considerably, you should be able to fit hundreds of rules in an Amiga with 512K of memory. Alternatively, Amiga BASIC allows you to set up deductive routines as separate programs that can be called as overlays when needed by the top-level routines, then deleted. The only precondition is that a called program has to be saved as an ASCII file (*SAVE "filespec",A*) or else a "bad file mode" error message appears.

You can call a secondary program with:

```
CHAIN MERGE "filespec",
   [expression],ALL,DELETE range
```

The *ALL* assures that all variables are shared between the main calling program and the called program, a fea-

The fastest, tightest code.



(Though the same can hardly be said of the name.)

We have to tell you, we had a hard time getting the name down this short.

Because Microsoft's new FORTRAN Compiler actually has a far longer list of features.

It uses the same optimizer and code generator technology that made our C Compiler the industry leader.

And we've also added special loop optimizations that give you the smallest, fastest FORTRAN code a PC can handle.

"Now Microsoft's FORTRAN Optimizing Compiler generates such fast code that an IBM PC/XT approaches the speed of the VAX."

Peter Osgood, MIT, Project Athena, Director of the Real Time Lab Project.

This compiler has already passed the toughest test there is. It's been

Microsoft FORTRAN Optimizing Compiler Version 4.0.

- ♦ Uses the Microsoft C optimizing technology, plus loop optimization to generate the fastest executable code for MS-DOS. NEW!

Execution Speed (in Seconds)	Microsoft FORTRAN v. 4.0	Ryan-McFarland FORTRAN v. 2.11	IBM Professional FORTRAN v. 1.22
Sieve	7.97	9.33	38.51
Whetstone	53.82	58.67	79.04
Lookup	5.82	18.61	26.02

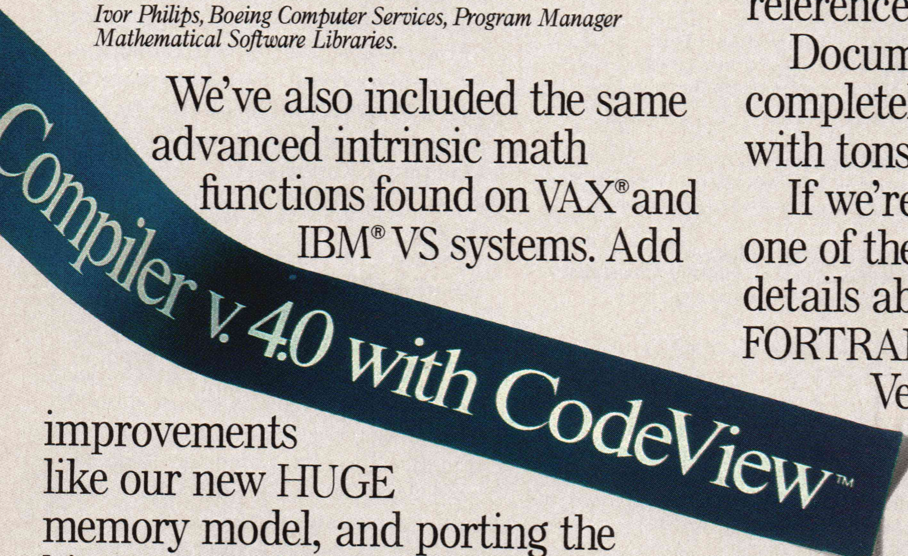
- ♦ Fully GSA certified for ANSI 77 compatibility with no errors at the highest level. NEW!
- ♦ Numerous IBM VS and DEC VAX extensions. NEW!
- ♦ Microsoft CodeView: Window-oriented source-level debugger. NEW!
  - —Debug using your original source code, the resulting disassembly or both intermingled.
  - —Watch and change the values of your local and COMMON variables as you debug.
  - —Set conditional breakpoints on variables, expressions or memory; trace and single step.
  - —Debug Microsoft C programs as well as Microsoft Fortran programs.
  - —Watch and change registers and flags as you execute.
  - —Easily debug graphics oriented programs since program output is kept separate from debugger output.

GSA-certified as Full ANSI FORTRAN 77, and 100% error-free.

"The Microsoft FORTRAN Optimizing Compiler let us port the 200,000 line Boeing Mathematical Library (BCSLIB) with virtually no changes. This ANSI FORTRAN 77 code was ported directly from Cray, CDC, DEC, IBM and other mainframes and workstations."

Ivor Philips, Boeing Computer Services, Program Manager Mathematical Software Libraries.

We've also included the same advanced intrinsic math functions found on VAX® and IBM® VS systems. Add



improvements like our new HUGE memory model, and porting the biggest mainframe programs has never been easier.

Among the many additions we've made to our package is our exclusive CodeView™ windowing debugger.

It lets you trace through programs at any level you want, from source code to assembly language.

You can open windows and watch both variables (local and COMMON) and CPU registers change.

You can set conditional breakpoints using variables and expressions.

Debugging gets even easier with the compiler's advanced diagnostics. Detailed error messages are thoroughly explained and cross-referenced in our new manuals.

Documentation that has been completely revised and expanded with tons of examples.

If we're talking your language, use one of the numbers below for more details about the Microsoft® ANSI FORTRAN 77 Optimizing Compiler Version 4.0 with CodeView, and the name of your nearest dealer.

(Even if the call's toll-free, it may be a good idea to refer to it as "FORTRAN 4.0" for short.)

Microsoft® FORTRAN

The High Performance Software.

Call (800) 426-9400. In Washington State or Alaska, (206) 882-8088. In Canada, (416) 673-7638.

- ♦ Medium, Large and Huge Memory Model Libraries. NEW!
- ♦ Mix models with NEAR, FAR and new HUGE pointers.
- ♦ Common blocks and arrays greater than 64K.
- ♦ Choose from three math libraries and generate in-line 8087/80287 instructions or floating point calls:
  - —floating point emulator (utilizes 8087/80287 if installed)
  - —8087/80287 coprocessor support
  - —alternate math package—extra speed without an 8087/80287
- ♦ Link your FORTRAN routines with Microsoft C (v. 4.0 or higher), Microsoft Pascal (v. 3.3 or higher) or Microsoft Macro Assembler.
- ♦ Largest number of 3rd party support libraries available.
- ♦ Provides more detailed diagnostic error messages (almost twice as many as competitors) and extensive documentation with non-ANSI 77 features highlighted. NEW!
- ♦ Proven reliability—tested with over 2.5 million lines of code compiled and executed.
- ♦ MS-DOS® network support with file / record locking and sharing.
- ♦ Microsoft Program Maintenance Utility rebuilds your applications after your source files have changed. NEW!
- ♦ Other utilities including faster overlay linker (links over 1Mbyte object code), library manager, EXE file compression utility, EXE file header utility, MS-DOS environment setting utility and setup utility.

EXPERT SYSTEMS
(continued from page 49)

ture that this kind of system requires. Using this technique in conjunction with a hard disk enables you to construct an expert system with thousands of facts and rules.

Such disk-intensive software would probably be quite slow, so the best thing to do is to keep the whole program in memory by shortening the variable and label names. Removal of the *Delay:* subroutine will also speed things up, showing how interpreted BASIC can hold its own against interpreted LISP, at least under these circumstances. Still, I'm waiting for the Amiga BASIC compiler to appear.

Availability

All the source code for articles in this issue (except for C Chest) is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

Notes

1. P. H. Winston and B. K. P. Horn, *LISP* (Reading, Mass.: Addison-Wesley, 1981).
2. Richard O. Duda and John G. Gaschnig, "Knowledge-Based Expert Systems Come of Age," *Byte*, vol. 6 no. 9 (September 1981): 238–281.
3. John Zarrella, *Language Translators* (Suisun City, Calif.: Microcomputer Applications, 1982).
4. J. B. Adams, "A Probability Model of Medical Reasoning and the MYCIN Model," *Mathematical Biosciences* 32 (1976): 177–186.

DDJ

(Listing begins on page 74.)

Vote for your favorite feature/article.
Circle Reader Service **No. 4.**

WINDOWS FOR DATA™

The first choice of professional C programmers

"Windows for Data is the best programming tool I've ever used.
It's the most flexible I've seen.
Whenever I've wanted to do something,
I've been able to find a way."

Steven Weiss,
Stratford Systems

Professionals choose our tools because they are designed, crafted, and supported for professionals. Here at Vermont Creative Software, we understand that performance and pleasure in programming derive from more than a long list of functions. **Windows for Data** provides:

PROFESSIONAL FLEXIBILITY: Our customers repeatedly tell us how they've used WFD in ways we never imagined - but which we anticipated by designing WFD for unprecedented adaptability. Virtually every capability and feature can be modified to meet special needs. You will be amazed at what you can do with WFD.

PROFESSIONAL PERFORMANCE: Screen output is crisp and fast. Windows, menus, and data-entry forms snap up and down from the screen. WFD is built upon and includes **Windows for C, the windowing system rated #1 in speed and overall quality** in PC Tech Journal (William Hunt, July 1985).

PROFESSIONAL RELIABILITY: An unreliable tool is worse than no tool at all. VCS products are known in the industry for their exceptional reliability. Ask anyone who owns one.

PROFESSIONAL DOCUMENTATION: Over 600 pages of documentation provide step-by-step explanations for each major application, a reference page for each function, listings of functions alphabetically and by usage, and a fully cross-referenced index. Extensive tutorials and demonstration programs assist learning.

PROFESSIONAL TECHNICAL SUPPORT: The same expert programmers that develop our products provide prompt, knowledgeable technical support.

PROFESSIONAL PORTABILITY: High-performance versions of VCS products are available for XENIX, UNIX, and VMS, as well as DOS. No royalties on end-user applications.

OUR CHALLENGE AND GUARANTEE

If you have an application where no other tool can do the job, try **Windows for Data.** If it doesn't help you solve your problem, RETURN FOR A FULL REFUND. YOU MUST BE SATISFIED.

Ask for **FREE DEMO DISKETTE**

Vermont Creative Software

21 Elm Ave.
Richford, VT 05476
Telex: 510-601-4160 VCSOFT
Tel.: 802-848-7738

Prices: PCDOS* $395; XENIX, VMS, UNIX Call.
*PCDOS specify C compiler.

WINDOWS FOR DATA

for DOS, UNIX, VMS ...

The complete windowing data entry, menu, and help system that does the hard job others can't — we **guarantee** it!

Pop-up data entry windows; field types for all C data types, plus decimals, dates, and times; auto conversion to and from strings for all field types; system and user supplied validation functions; range checking; required, must-fill, and protected fields; free-form movement; multiple-choice field entry; scrollable sub-forms. Branch and nest windows, forms, and menus.

Complete context-sentitive help system with pop-up windows and scrollable text.

Pop-up, pull-down, scrollable, and Lotus-style menus.

NEW FOR DEBUGGING: Exclusive **VCS Error Traceback System** automatically identifies the location and cause of program errors. Eliminates the need to code error checks on all function calls! **VCS Memory Integrity Checking** helps catch those hard-to-detect, memory-corruption errors.

NEW FOR ERROR HANDLING: Install your own error handler to be called whenever a function detects an error.

NEW FORM LAYOUT UTILITY simplifies form design.

THE PROGRAMMER'S SHOP

helps save time, money and cut frustrations. Compare, evaluate, and find products.

RECENT DISCOVERY

Turbo Expert by Thinking Technologies - Menu driven expert system generation package details reasoning, comes with tutorial, manual, demos. "Corporate" supports up to 4000 rules. "Startup" up to 400. Corporate $359, PC Startup $129

AI-Expert System Dev't

Arity System - use with C	MS	$ 259
Auto-Intelligence	PC	$ 749
Experteach - Powerful, samples	PC	$ 349
Exsys	PC	$ 309
Runtime System	PC	$ 479
Insight 2+	MS	$ 379
Intelligence/Compiler	PC	$ 749
SQL Dev't Package	MS	$ 259
Texas Instruments:		
PC Easy	PC	$ 439
Personal Consultant Plus	PC	$2599

AI-Lisp

Microsoft MuLisp 85	MS	$ 179
PC Scheme LISP - by TI	PC	$ 85
TransLISP - learn fast	MS	Call
TransLISP PLUS		
Optional Unlimited Runtime		$ 150
PLUS for MSDOS		$ 179

Others: IQ LISP ($155), IQC LISP ($269).

AI-Prolog

APT - Active Prolog Tutor - build applications interactively	PC	Call
ARITY Standard - full, 4 Meg		
Interpreter - debug, C, ASM	PC	$ 309
COMPILER/Interpreter-EXE	PC	$ 699
With Exp Sys, Screen - KIT	PC	$1129
Standard Prolog	MS	$ 79
MacProlog Complete	MAC	$ 295
MicroProlog - intro	MS	$ 85
MicroProlog Prof.	MS	$ 339
MPROLOG P550	PC	$ 175
Prolog-86 - Learn Fast	MS	$ 89
Prolog-86 Plus - Develop	MS	$ 229
TURBO PROLOG by Borland	PC	$ 69

AI-Other

Q'NIAL - APL with LISP.	PC	$ 349
Smalltalk-80 - Xerox improved	PC	$ 995
Smalltalk/V-graphics	PC	$ 89

Atari ST & Amiga

Manx, Lattice, & Metacomco.		Call
Amiga - LINT by Gimpel	Amiga	$ 79
Cambridge LISP	Amiga	$ 200
Lattice C	ST, Amiga	$ 139
Lattice Text Utilities	Amiga	$ 75
Megamax - tight, full	ST	$ 200

FEATURES

r-tree - report generation for ctree. Multiple file handling, fixed or variable length. Many built-in functions like Boolean, computational functions, string, date handling, numeric to string conversion. Source in C. PC $ 249

Advantage C++ - Object-oriented enhancements for most major C compilers also maintain compatibility with existing C code. Write reusable easily maintainable code, develop large applications with fewer bugs. Call

Free Literature Compare Products

Evaluate products. *Compare* competitors. Learn about new alternatives. One *free call* brings information on just about any programming need. Ask for any ***"Packet"*** or Addon Packet □ AI □ ADA, Modula □ BASIC □ "C" □ COBOL □ Editors □ FORTH □ FORTRAN □ PASCAL □ UNIX/PC or □ Debuggers, Linkers.

Our Services:

- **Programmer's Referral List**
- **Compare Products**
- **Help find a Publisher**
- ***Evaluation Literature FREE***
- **BBS - 7 PM to 7 AM 617-826-4086**
- **Dealers Inquire**
- **Newsletter**
- **Rush Order**
- **Over 700 products**
- **National Accounts Center**

Basic

Basic Development System	PC	$ 105
Basic Development Tools by Sterling Castle	PC	$ 89
Basic Windows by Syscom	PC	$ 95
BetterBASIC	PC	$ 129
8087 Math Support	PC	$ 75
Run-time Module	PC	$ 169
Better Tools - for Better Basic	PC	$ 95
CADSAM FILE SYSTEM-full	MS	$ 69
Finally - by Komputerwerks	PC	$ 85
GoodBas - maintain code	PC	$ 95
LPI Basic - MS compatible	UNIX	$1100
Prof. Basic - Interactive, debug	PC	$ 75
8087 Math Support	PC	$ 45
QuickBASIC	PC	$ **69**
TRUE Basic - ANSI	PC	$ 119
Run-time Module	PC	$ 129
Turbo BASIC - by Borland	PC	$ 69

Cobol

Macintosh COBOL - full	MAC	$ 459
MBP - Lev. II, native	MS	$ 819
Microfocus Professional Cobol	PC	$2295
VS Workbench	PC	$3379
Microsoft COBOL	MS	$ 439
Microsoft Cobol Tools	PC	$ 209
Realia - very fast	MS	$ 819
Ryan McFarland COBOL	MS	Call
COBOL-8X	MS	Call
Screenplay-by Flexus. Interactive screen mgmt.	PC	$ 175

Editors for Programming

BRIEF Programmer's Editor	PC	Call
EMACS by UniPress Source: $929		$ 299
Epsilon - like EMACS, full C-like language for macros.	PC	$ 155
KEDIT - like XEDIT	PC	$ 99
Lattice Screen Editor - multiwindow, multitasking Amiga $ 89	MS	$ 109
Micro Focus Micro/SPF	PC	$ 49
PC/EDT - macros	PC	$ 250
PC/VI - by Custom Software	MS	$ 109
Personal REXX	PC	$ 99
PMATE - power, multitask	PC	$ 119
SPF/PC - fast, virtual memory	PC	$ 139
Vedit	MS	$ 107
Vedit PLUS	MS	$ 139

C Libraries-Communications

Asynch by Blaise	PC	$ 135
Essential Comm Library	PC	$ 135
With Debugger	PC	$ 199
Greenleaf Comm Library	PC	$ 129
Multi-Comm - add multitasking, use w/Multi-C	PC	$ 149

RECENT DISCOVERY

VXM by Command Technologies - Allows intelligent program control sharing by different environments like PCDOS-VAX. "Software Robots" use resources as needed, convert formats automatically, transmit data. Innovative mini-micro integration. PC Call

C Language-Compilers

AZTEC C86 - Commercial	PC	$499
Datalight C - fast compile, good code, 4 models, Lattice compatible, Lib source. Dev'rs Kit	PC	$ 77
Datalight Optimum - C	MS	$ 99
with Light Tools by Blaise	PC	$168
Lattice C - from Lattice	MS	$275
Mark Williams - w/debugger	MS	$369
Let's C Combo Pack	PC	$ 99
Let's C	PC	$ 59
Microsoft C 4.0- Codeview	MS	**$279**
Uniware Cross Assemblers	MS	$249
Cross Dev't Tools	MS	Call
Rex - C/86 by Systems & Software - standalone	MS	$695
Wizard C	MS	$359
Rom Development Package	MS	$299

C Language-Interpreters

C-terp by Gimpel - full K & R	MS	$229
C Trainer - by Catalytix	PC	$ 89
INSTANT C - Source debug, Edit to Run-3 seconds, .OBJs	MS	$379
Interactive C by IMPACC Assoc.	PC	$209
Introducing C-self paced tutorial	PC	$105
Run/C Professional	MS	$169
Run/C Lite	MS	$ 89

C Libraries-General

Blackstar C Function Library	PC	$ 79
C Essentials - 200 functions	PC	$ 83
C Function Library	MS	$119
C Tools Plus (1 & 2) - Blaise	PC	$135
C Utilities by Essential	PC	$137
C Worthy Library - Complete, machine independent	MS	$249
Entelekon C Function Library	PC	$119
Entelekon Superfonts for C	PC	$ 45
Greenleaf Functions-portable, ASM		$139
LIGHT TOOLS by Blaise	PC	$ 69

C Libraries-Files

FILES: C Index by Trio - full B+ Tree, vary length field, multi compiler		
/File is object only	MS	$ 89
/Plus is full source	MS	$319
CBTREE - Source, no royalties	MS	$ 99
CTree by Faircom - no royalties	MS	$319
rtree - report generation	PC	$249
dbQUERY - ad Loc, SQL - based	MS	$159
dbVISTA - full indexing, plus optional record types, pointers, Network.		
Object only - MS C, LAT,	C86	$145
Source - Single user	MS	$399
Source - **Multiuser**	MS	$799
dBx - translator	MS	$315
w/source to library	MS	$349

FEATURE

Sapiens V8 - virtual memory management for C programmers on PC's provides 8M workspace, 64-bit emulation, virtual stack library and heap. Link to MS, Lattice, Aztec. PC $300

We support MSDOS (not just compatibles), PCDOS, Xenix-86, CPM-80, Macintosh, Atari ST, and Amiga.

Listing One *(Text begins on page 16.)*

```
#define PGM_ID "SILOAM CI-C86 Ver. of 11/22/86 for PC-DOS 2.x+"

/*     An Adaptive Template Matching Image Categorizer
 *        (An Experimental Computer Vision Program)
 *
 *      This program implements a trainable pattern classifier as
 *  a committee network of threshold logic units.  It learns to
 *  recognize patterns by being trained from a set of prototype
 *  patterns presented in a training file.  The training file is
 *  organized as a set of visual images represented as an orthogonal
 *  array of picture elements, or pixels.  Each pixel is a number
 *  representing the gray-scale value of that point in the image.
 *  Associated with each pattern is a number, or tag, that
 *  represents the category to which that pattern belongs.
 *
 *        R. J. Brown
 *        Elijah Laboratories International
 *        5225 N.W. 27th Court
 *        Margate, FL 33063
 *        (305) 979-1567
 *
 *   Ownership: I hereby place this program in the public domain.
 *
 *   System:    Red River ATlas 10 MHz 80286 IBM-PC/AT clone
 *
 *   Compiler:  C86 Version 2.30H; Computer Innovations, Inc.
 *
 */

#include "stdio.h"              /* needed for stream input/output       */

#define FALSE      0            /* boolean constant for 'false'         */
#define TRUE      !FALSE        /* boolean constant for 'true'          */

#define NULL ((int *)0)         /* the pointer to nowhere               */

#define void                    /* function that returns no value       */

#define forall(index,limit)\
        for((index)=0;(index)<(limit);(index)++)     /* looping word    */

#define kase(id,stmt)    \
        case(id): {      \
            stmt;        \
            break;       \
            }           /* shorthand form for case statement        */

#define u(x)  ((unsigned)(x))   /* shorthand for '(unsigned)' cast      */

typedef unsigned char  byte;    /* an 8-bit byte of storage             */
typedef unsigned int   word;    /* a 16-bit word of storage             */

typedef word     boolean;       /* a decision variable,
                                 * 'true' or 'false value only */

typedef ELTYPE     element;     /* an element is a real number          */
typedef DOTYPE     DOT;         /* type of a dot product may be bigger! */
typedef element   *vector;      /* a vector is a set of elements        */

typedef vector    tlu;          /* a tlu is a vector                    */

typedef struct {                /* the collection of                    */
        tlu      *wtpt;         /* a set of tlu weight points,          */
        DOT      *dot;          /* and dot product save cells           */
        }          committee;   /* is a committee                       */

typedef char      *pointer;     /* a general pointer to whatever...     */

/*************************************************************************
 *
 *        G l o b a l   V a r i a b l e   D e f i n i t i o n s
 *
 *************************************************************************/

FILE    *pat,           /* the input training pattern file              */
        *fopen();       /* the file opener                              */
byte    patname[64],    /* ascii filename of input file                 */
        *index();       /* string search library function               */
```

```
int      ncom,            /* number of committees in the network          */
         patwide,         /* pattern width in pixels                      */
         pathite,         /* pattern height in pixels                     */
         pats_so_far,     /* how many patterns in file so far             */
         pats_missed,     /* how many patterns were mis-recognized so far */
         missed,          /* # of patterns missed on this pass            */
         tlus_trained,    /* how many tlu's have been adjusted so far     */
         npass,           /* number of current pass thru pattern file     */
         log_level,       /* level of detail for run-time logging         */
         dim,             /* number of elements in a vector (dimension)   */
         ntlu,            /* number of tlus per committee                 */
         corr_incr,       /* fixed increment correction constant          */
         *vote;           /* pointer to vote count array                  */

boolean goofed,           /* mis-recognition indicator for training loop  */
         start_over,      /* select start over on error training strategy */
         absolute,        /* flag for absolute correction training method */
         *decsn,          /* pointer to network's decision array          */
         *class;          /* pointer to class (category) array            */


DOT      patmag;          /* pattern magnitude (used for training)        */

element  fraction,        /* correction fraction for training             */
         maxel=0;         /* maximum element in a weight point            */
         radius;          /* average radius (distance from origin)
                           * of tlu weight point at initialization */

vector   pattern;         /* pointer to current input pattern             */

committee *net;           /* pointer to network as an array of committees */


/**************************************************************************
 *
 *                Library   Routines
 *
 **************************************************************************/

extern float   atof();  /* ascii to float library conversion routine     */
extern double  sqrt();  /* square root library function                  */
extern pointer calloc();/* memory allocation library function            */
extern long    time();  /* benchmark timing routine                      */


/**************************************************************************
 *
 *     B A N N E R  --   D i s p l a y   P r o g r a m   I. D.
 *
 **************************************************************************/

void banner() { /* display program identification information            */

    printf("\n%s",PGM_ID);       /* Program Identification is #define'd
                                  * at top of source file */

    printf("\nWritten by:  R. J. Brown, Elijah Laboratories Intn'l");
    printf("\nThis program is in the Public Domain.\n");
}


/**************************************************************************
 *
 *           H E L P   D i s p l a y   S c r e e n
 *
 **************************************************************************/

void help() {    /* some user friendly help for the uninitiated !          */

    printf("Simple Image Learning On Adaptive Machinery\n");
    printf("An Adaptive Template Matching Image Categorizer\n");
    printf("\n");
    printf("    R. J. Brown, Elijah Laboratories International\n");
    printf("    5150 W. Copans Rd. Suite 1135, Margate FL 33063\n");
    printf("\n");
    printf("usage:        siloam <options> filename[.ext]\n\n");
    printf("where:  filename -- is the input pattern file.\n\n");
    printf("options:   -r##.# -- gives initialization radius.\n");
    printf("             -t## -- gives number of TLUs per committee.\n");
    printf("               -o -- start over on error,\n\n");
    printf("choose one: -i## -- fixed increment correction, ## = incr.\n");
    printf("               -a -- absolute correction.\n");
```

(continued on next page)

Listing One *(Listing continued, text begins on page 16.)*

```
    printf("          -f##.# -- fractional correction, ##.# is lambda.\n");
    printf("             -l# -- logging level: 0=least; 3=most.\n");
    exit(0);
}


/***********************************************************************
 *
 *     S I G N  --  T h e   S i g n   O f   A n   E l e m e n t   +/- 1
 *
 ***********************************************************************/

int sign(x)       /* return the sign of a number as plus or minus one     */
element x;        /* argument is an element                               */
{
    return( x<(element)0 ? -1   /* if number is negative, return -1   */
                     :  1 );/* else return +1                         */
}


/***********************************************************************
 *
 *   I S I G N  --  T h e   S i g n   O f   A n   I n t e g e r   +/- 1
 *
 ***********************************************************************/

int isign(x)      /* return the sign of a number as plus or minus one     */
int x;            /* argument is an integer                               */
{
    return( x<0  ? -1           /* if number is negative, return -1   */
                 :  1 );        /* else return +1                     */
}


/***********************************************************************
 *
 *  A B S  --  A b s o l u t e   V a l u e   O f   A n   E l e m e n t
 *
 ***********************************************************************/

element eabs(x)       /* the absolute value of an element                 */
element x;            /* argument is an element                           */
{
    return( x<0 ? -x       /* if number is negative, make it positive    */
                : x );     /* else return it like it is                  */
}


/***********************************************************************
 *
 *  I A B S  --  A b s o l u t e   V a l u e   O f   A n   I n t e g e r
 *
 ***********************************************************************/

int iabs(x)            /* the absolute value of an integer                */
int x;                 /* argument is an integer                          */
{
    return( x<0 ? -x         /* if number is negative, make it positive  */
                : x );       /* else return it like it is                */
}

/***********************************************************************
 *
 *              A L P H A  --  S t e p   F u n c t i o n
 *
 ***********************************************************************/

int alpha(x)          /* step function return zero or one                 */
int x;                /* argument is an integer (in this program...)      */
{
    return( x>0 ? 1         /* if argument strictly positive, return one */
                : 0 );      /* else return zero                          */
}


/***********************************************************************
 *
 *        M O V E  --  S t r i n g   M o v e   F u n c t i o n
 *
 ***********************************************************************/
```

(continued on page 60)

SAS Institute Inc. Announces

Lattice C Compilers for Your IBM Mainframe

Two years ago...

SAS Institute launched an effort to develop a subset of the SAS® Software System for the IBM Personal Computer. After careful study, we agreed that C was the programming language of choice. And that the Lattice® C compiler offered the quality, speed, and efficiency we needed.

One year ago...

Development had progressed so well that we expanded our efforts to include the entire SAS System on a PC, written in C. And to insure that the language, syntax, and commands would be identical across all operating systems, we decided that all future versions of the SAS System—regardless of hardware—would be derived from the same source code written in C. That meant that we needed a C compiler for IBM 370 mainframes. And it had to be good, since all our software products would depend on it.

So we approached Lattice, Inc. and asked if we could implement a version of the Lattice C compiler for IBM mainframes. With Lattice, Inc.'s agreement, development began and progressed rapidly.

Today...

Our efforts are complete—we have a first-rate IBM 370 C compiler. And we are pleased to offer this development tool to you. Now you can write in a single language that is source code compatible with your IBM mainframe and your IBM PC. We have faithfully implemented not only the language, but also the supporting library and environment.

Features of the Lattice C compiler for the 370 include:

- **Generation of reentrant object code.** Reentrancy allows many users to share the same code. Reentrancy is not an easy feature to achieve on the 370, especially if you use non-constant external variables, but we did it.
- **Optimization of the generated code.** We know the 370 instruction set and the various 370 operating environments. We have over 100 staff years of assembler language systems experience on our development team.
- **Generated code executable in both 24-bit and 31-bit addressing modes.** You can run compiled programs above the 16 megabyte line in MVS/XA.
- **Generated code identical for OS and CMS operating systems.** You can move modules between MVS and CMS without even recompiling.
- **Complete libraries.** We have implemented all the library routines described by Kernighan and Ritchie (the informal C standard), and all the library routines supported by Lattice (except operating system dependent routines), plus extensions for dealing with 370 operating environments directly. Especially significant is our byte-addressable Unix®-style I/O access method.
- **Built-in functions.** Many of the traditional string handling functions are available as built-in functions, generating in-line machine code rather than function calls. Your call to move a string can result in just one MVC instruction rather than a function call and a loop.

In addition to mainframe software development, you can also use our new cross-compiler to develop PC software on your IBM mainframe. With our cross-compiler, you can compile Lattice C programs on your mainframe and generate object code ready to download to your PC.

With the cross-compiler, we also offer PLINK86™ and PLIB86™ by Phoenix Software Associates Ltd. The Phoenix link-editor and library management facility can bind several compiled programs on the mainframe and download immediately executable modules to your PC.

Tomorrow...

We believe that the C language offers the SAS System the path to true portability and maintainability. And we believe that other companies will make similar strategic decisions about C. Already, C is taught in most college computer science curriculums, and is replacing older languages in many. And almost every computer introduced to the market now has a C compiler.

C, the language of choice...

C supports structured programming with superior control features for conditionals, iteration, and case selection. C is good for data structures, with its elegant implementation of structures and pointers. C is conducive to portable coding. It is simple to adjust for the size differences of data elements on different machines.

Continuous support...

At SAS Institute, we support all our products. You license them annually; we support them continuously. You get updates at no additional charge. We have a continuing commitment to make our compiler better and better. We have the ultimate incentive—all our software products depend on it.

For more information...

Complete and mail the coupon today. Because we've got the development tool for your tomorrow.

SAS®

SAS Institute Inc.
SAS Circle, Box 8000
Cary, NC 27511-8000
Telephone (919) 467-8000 x 7000

I want to learn more about:

☐ the C compiler for MVS software developers
☐ the C compiler for CMS software developers
☐ the cross-compiler with PLINK86 and PLIB86

today...so I'll be ready for tomorrow.

Please complete or attach your business card.

Name ______________________

Title ______________________

Company ______________________

Address ______________________

City ______________ State ________ ZIP ________

Telephone ______________________

Mail to: SAS Institute Inc., Attn: CC, SAS Circle, Box 8000, Cary, NC, USA. 27511-8000. Telephone (919) 467-8000, x 7000

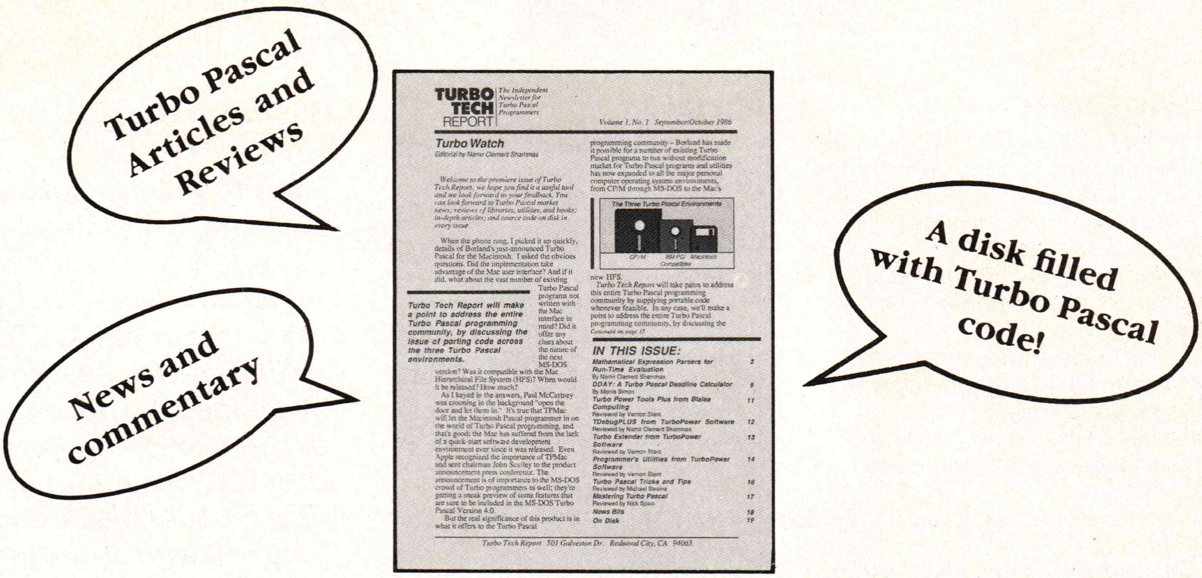
DDJ 4/87

Listing One *(Listing continued, text begins on page 16.)*

```
char *move(src,dst)        /* move a string returning ptr to end of result */
char *src,*dst;            /* pointers to source & destination strings     */
{
    while(0!=((*dst++)=(*src++)));  /* copy bytes until end of source    */
    return(--dst);                  /* return ptr to end of destination */
}


/**********************************************************************
 *
 *    R A D I U S    S T A T I S T I C S -- S u m m a r y   I n f o
 *
 **********************************************************************/

void radius_statistics() {    /* show how weight points are distributed  */

    element r,            /* current radius accumulator                    */
            *pe;          /* pointer to current element                    */
    float   mu=0,         /* mean of radii                                 */
            sigma=0;      /* standard deviation of radii                   */

    committee *pc=net;    /* pointer to current committee                  */

    vector  *pt;          /* pointer to current tlu                        */

    int     c,                /* committee loop counter                    */
            t,                /* tlu loop counter                          */
            e,                /* element loop counter                      */
            n=ncom*ntlu;      /* number of tlu's altogether                */

    forall(c,ncom) {              /* for all committees...                 */
        pt=pc++->wtpt;                /* point to first tlu                */
        forall(t,ntlu) {              /* for all tlu's...                  */
            pe=*pt++;                     /* point to first element        */
            r=0.;                         /* initialize radius tally       */
            forall(e,dim) {               /* for all elements...           */
                r+=(*pe)*(*pe);               /* accumulate radius sqr'd  */
                pe++;                         /* point to next element    */
            }
            mu+=sqrt((float)r);           /* accumulate sum of radii       */
            sigma+=(float)r;              /* accumulate variance variable */
        }
    }
    mu/=(float)n;                 /* divide to get overall average radius */
    sigma-=mu*mu*n;               /* compute variance                      */
    sigma=sqrt(sigma)/mu;         /* compute standard deviation            */

    printf("\nmean of the radii:  %f",mu);       /* print statistical    */
    printf("\nstandard deviation: %f",sigma);    /* summary of weight    */
    printf("\n");                                /* point distribution   */
}


/**********************************************************************
 *
 *     R E A D   H E A D E R -- R e a d   F i l e   H e a d e r
 *
 **********************************************************************/

void read_header()          /* read training file header information    */
{

    rewind(pat);                    /* rewind pattern file                */
    pats_so_far=0;                  /* reset pattern sequence counter     */

    fscanf(pat,                     /* header comes from pattern file     */

           "hdr %d %d %d \n",       /* header must start with 'hdr'
                                     * then read header information
                                     * composed of three numbers */

           /* put this information into the following global variables */

           &ncom,                   /* number of committees in network   */
           &patwide,                /* pattern width in pixels           */
           &pathite);               /* pattern height in pixels          */

}
```

(continued on page 65)

Turbo Tech Report Speaks Your Language.



TURBO TECH REPORT | The Independent Newsletter for Turbo Pascal Programmers

Volume 1, No. 1 September/October 1986

Turbo Watch

Editorial by Namir Clement Shammas

Welcome to the premiere issue of Turbo Tech Report; we hope you find it a useful tool and we look forward to your feedback. You can look forward to Turbo Pascal market news; reviews of libraries, utilities, and books; in-depth articles; and source code on disk in every issue

When the phone rang, I picked it up quickly, details of Borland's just-announced Turbo Pascal for the Macintosh. I asked the obvious questions. Did the implementation take advantage of the Mac user interface? And if it did, what about the vast number of existing Turbo Pascal programs not written with the Mac interface in mind? Did it offer any clues about the nature of the next MS-DOS version? Was it compatible with the Mac Hierarchical File System (HFS)? When would it be released? How much?

Turbo Tech Report will make a point to address the entire Turbo Pascal programming community, by discussing the issue of porting code across the three Turbo Pascal environments.

As I keyed in the answers, Paul McCartney was crooning in the background "open the door and let them in." It's true that TPMac will let the Macintosh Pascal programmer in on the world of Turbo Pascal programming, and that's good; the Mac has suffered from the lack of a quick-start software development environment ever since it was released. Even Apple recognized the importance of TPMac and sent chairman John Sculley to the product announcement press conference. The announcement is of importance to the MS-DOS crowd of Turbo programmers as well; they're getting a sneak preview of some features that are sure to be included in the MS-DOS Turbo Pascal Version 4.0.

But the real significance of this product is in what it offers to the Turbo Pascal programming community – Borland has made it possible for a number of existing Turbo Pascal programs to run without modification market for Turbo Pascal programs and utilities has now expanded to all the major personal computer operating system environments, from CP/M through MS-DOS to the Mac's new HFS.

The Three Turbo Pascal Environments: CP/M, IBM PC Compatibles, Macintosh

Turbo Tech Report will take pains to address this entire Turbo Pascal programming community by supplying portable code whenever feasible. In any case, we'll make a point to address the entire Turbo Pascal programming community, by discussing the

Continued on page 15

IN THIS ISSUE:

Mathematical Expression Parsers for Run-Time Evaluation By Namir Clement Shammas	2
DDAY: A Turbo Pascal Deadline Calculator By Morris Simon	9
Turbo Power Tools Plus from Blaise Computing Reviewed by Vernon Stant	11
TDebugPLUS from TurboPower Software Reviewed by Namir Clement Shammas	12
Turbo Extender from TurboPower Software Reviewed by Vernon Stant	13
Programmer's Utilities from TurboPower Software Reviewed by Vernon Stant	14
Turbo Pascal Tricks and Tips Reviewed by Michael Swaine	16
Mastering Turbo Pascal Reviewed by Nick Spain	17
News Bits	18
On Disk	19

Turbo Tech Report 501 Galveston Dr. Redwood City, CA 94063

The newsletter/disk publication for Turbo Pascal® users

Are you a devoted Turbo Pascal programmer, tired of reading about other languages? Are you looking for powerful utilities written in Turbo Pascal that you can use to develop software or incorporate into your programs? Are you interested in improving and expanding your Turbo Pascal programming skills?

Then you deserve a subscription to *Turbo Tech Report*, the bimonthly newsletter/disk publication from the publishers of *Dr. Dobb's Journal* and *Micro/Systems Journal*. Each issue delivers more than 250K of Turbo Pascal source code programs on disk, and 20+ pages of articles, Turbo Pascal software and book reviews, and analysis and commentary. It's the only publication delivering such focused technical articles with code on disk—and it doesn't waste your time with information about other programming languages. Each valuable issue contains:

• ***Articles*** on topics like speedy 3D graphics, mathematical expression parsers, creating global gotos, memory resident and AI applications and more—all written by Turbo experts.

• ***Reviews*** of the latest Turbo Pascal software programs from companies like Borland International, Blaise Computing, Media Cybernetics, Nostradamus, TurboPower Software, and more!

• ***News and commentary*** detailing the latest products and developments in the Turbo Pascal programming community.

• ***A disk filled with Turbo Pascal code!*** You'll get the Turbo Pascal utilities and routines discussed in the newsletter's articles, as well as applications developed by Turbo users from around the world. You'll receive programs that make labels, generate menus, provide faster screen access, transfer files between CP/M and MS-DOS computers, and more!

If you're an expert Turbo Pascal programmer or a novice interested in expanding your Turbo skills, you need a publication that speaks your language: *Turbo Tech Report*. Subscribe today at the special price of just $99—that's 33% off the regular price of $150. To order by credit card, call toll-free 1-800-528-6050 ext. 4001 and ask for item 300. Or mail the attached coupon with your payment to *Turbo Tech Report*, 501 Galveston Drive, Redwood City, CA 94063.

WE JUST GOT MORE SOPHISTICATED SO YOU CAN GET MORE BASIC.

We invented BASIC over 20 years ago.

Later, we re-invented it for micros as the True BASIC™ structured-programming language.

And the idea was: To make programming as easy and natural as possible. So you could concentrate on what to program. Not how.

Now there's True BASIC Version 2.0 for the IBM® PC and compatibles. Faster, more powerful and sophisticated than the original.

MORE GRAPHICS.

Right from the start, True Basic gave you terrific device-independent graphics. Built-in 2-D transforms. And support for multiple windows.

Now we've added more graphics and full mouse support.

So for the first time, you can create one program that will do superb graphics on CGA, EGA or Hercules displays. Without worrying about additional drivers or overlays. And on the EGA, you can SET COLOR MIX to define your own colors. Use four shades of blue if you want (and make our competitors green with envy).

MORE CONTROL.

We always supported you with recursion, local and global variables and separately compiled libraries.

Now you can have *modules*, too, the industrial-strength tool for building large applications.

Using modules makes it easier for you to share data between routines. Build data structures. Then, if you want, hide them from other parts of the program. So you can always be free to focus on the task at-hand.

Modules have their own initialization sections, so you can set up global variables or turn on instrumentation.

And, like other procedures in True BASIC, modules can be compiled separately and stored in a library where they can be shared by several applications. Or they can be loaded directly into the True BASIC environment as part of your customized workspace. So when you use True BASIC interactively, the modules look like built-in functions.

Modules made Modula-2 the successor to Pascal. Now they've put True BASIC one-up on all other BASICs.

MORE SPEED.

2.0 is 20 to 200 percent faster than True BASIC Version 1.0. Both compile times and execution speeds. And on some real-world benchmarks, we're faster than many native-code compilers.

MORE POWER.

Start with a complete matrix algebra package.

Then, since we support the use of 640K for both code and data, add arrays as large as you want.

Our compiled code is more compact than what other compilers generate, so there's more memory left for your application.

We've enhanced our dynamic array redimensioning and improved our built-in 8087/80287 support, making True BASIC the most powerful number-crunching BASIC around.

And if it's strings you crunch, we've added new string functions and raised the limit. So strings can be up to 64K characters long.

MORE DEBUGGING.

We pioneered breakpoints and immediate-mode capability in a compiled BASIC environment.

Now we've added utilities that allow you to visually TRACE through your program, and check the values of selected variables. Or print a cross-referenced listing.

Circle **no. 344** on reader service card.

And new compiler options like NO LET and NO TYPO let you decide how strictly you want your variable names checked.

MORE INNOVATION.

True BASIC has always had features like full-screen, scrollable editing. Block copy and block moves. And global search and replace.

Now, 2.0 keeps you on the leading edge of editing and file-management technology. With SCRIPT, to write the True BASIC equivalent of a DOS batch file. ECHO, to transfer your output to disk or printer. And ALIAS, to give you and your programs a better roadmap to your subdirectories.

There's also Version 2.0 of the Developer's Toolkit. With support for DOS interrupts. Pop-up menus. Even designer fonts.

And remember: your programs are portable to the other machines we support: the Apple Macintosh™ and Commodore Amiga.®

MORE SUPPORT.

Call your local dealer. Call us TOLL-FREE at 1-800-TR-BASIC. Or write to: True BASIC, Inc., 39 South Main Street, Hanover, NH 03755. We'll send you more information. Including a free demo disk.

See for yourself. That we're still true to our basic idea.

True BASIC™ inc.

Listing One *(Listing continued, text begins on page 16.)*

```
/**********************************************************************
 *
 *  R A N D O M  --  R a n d o m   N u m b e r   G e n e r a t o r
 *
 **********************************************************************/

element random() {        /* generate a uniformly distributed
                           * random number from the open interval (0...1) */

    return(rand()/16384.);         /* return scaled random integer        */
}


/**********************************************************************
 *
 *    I N I T   V A L -- I n i t i a l   E l e m e n t   V a l u e
 *
 **********************************************************************/

element init_val(radius)    /* generate init'l value for element a tlu  */
element radius;             /* the avarage radius of a weight point     */
{

    return(                                      /* return the          */
        (radius*sqrt(3.))/(sqrt((float)dim))     /* average weight value */
        * (2.*random()-1)                        /* scaled randomly by a */
    );                                           /* uniform distribution */
}


/**********************************************************************
 *
 *  I N I T I A L I Z E -- A l l o c a t e   S t o r a g e,  E t c.
 *
 **********************************************************************/

void initialize() {       /* allocate & initialize network array storage  */

    committee *pc;        /* pointer to current committee of network       */
    tlu       *pt;        /* pointer to current tlu of committee           */
    element   *pe,        /* pointer to current element of tlu             */
               x;         /* current initialization weight value           */

    int c,                /* committee index in network                    */
        t,                /* tlu index in committee                        */
        e;                /* element index in tlu                          */

    printf("\ninitializing");   /* say what's taking so long !             */

    dim=patwide*pathite+1;      /* number of elements in a tlu             */

    pattern=(vector)calloc(u(dim),           /* allocate the pattern       */
                   u(sizeof(element)));      /* vector                     */

    class=(boolean *)calloc(u(ncom),     /* allocate the class array       */
                u(sizeof(boolean)));     /* which will contain the
     * desired decision bits from the committees, as read from the
     * training file.  the actual verdict of the network will be
     * compared with this to see if training is required. */

    vote=(int *)calloc(u(ncom),       /* allocate the votes array          */
               u(sizeof(int)));       /* which will contain the
                                       * count of votes for each
                                       * committee. */

    decsn=(boolean *)calloc(u(ncom),     /* allocate the decision          */
                u(sizeof(boolean)));     /* array which will contain
                                          * the bits of the answer,
                                          * one bit per committee. */

    pc=net=(committee *)calloc(u(ncom),      /* allocate the network       */
                 u(sizeof(committee)));      /* as an array of committees */

    forall(c,ncom) {              /* for all committees in the network... */

        pc->wtpt=pt=(tlu *)calloc(u(ntlu),  /* allocate a committee        */
                          u(sizeof(tlu))); /* as an array of tlu's        */
        pc++->dot=(DOTYPE *)calloc(u(ntlu),      /* together with dot      */
                        u(sizeof(DOT)));         /* product save cells     */

        forall(t,ntlu) {          /* for all tlu's in the committee...    */
```

(continued on next page)

Listing One *(Listing continued, text begins on page 16.)*

```
        pe=*pt++=(element *)calloc(u(dim),        /* allocate a tlu     */
                          u(sizeof(element)));    /* as an array
                                                   * of elements */

        forall(e,dim) {                           /* for each weight...    */
            if(radius==0) *pe++=(e!=0);           /* grow connections?     */
            else {                                /* or adjust weights?    */
                x=eabs(*pe++=init_val(            /* adjust, get initial   */
                             (element)radius));/* weight value            */
                if(x>maxel) maxel=x;              /* update max magnitude */
            }
        }
  /* initialize each element to a random value such that the average
   * radius, or distance from the origin, of each weight point is 'radius'.
   * this will produce a distribution of weight points clustered near the
   * surface of a hyper-sphere as the starting condition. If the radius is
   * zero, then all weights will be set to zero except for the threshold
   * setting weight.  This is analogous to forcing the program to grow new
   * interneural connections on an as-needed basis, supposedly just like
   * the real brain does! */
        }
    }
    printf("\n");       /* perform new-line when initialize is done     */
}


/***********************************************************************
 *
 *     D O T P R O D   --   F o r m   A   D o t   P r o d u c t
 *
 ***********************************************************************/

DOT dotprod(x,y)              /* form the scalar product of two vectors   */
vector  x,y;                  /* both arguments are vectors               */
{
    DOT z=0;                  /* result accumulator, initialized to zero */
    int i;                    /* element index, used as loop counter      */

    forall(i,dim)             /* for all elements in each vector...       */
        z+=(*x++)*(*y++);             /* compute the dot product          */

    return(z);                /* return it to the caller                  */
}


/***********************************************************************
 *
 *   R E A D   C L A S S  --  R e a d   T h e   C l a s s   T a g
 *
 ***********************************************************************/

boolean read_class() {        /* read the class tag number for the image  */

    int        i,             /* loop counter for index in class array    */
               tmp;           /* temp cell to hold decimal category       */
    boolean    *pcl=class;    /* pointer to class (category) array        */

    if(fscanf(pat,"%d",&tmp)!=1)      /* read the pattern category        */
        return(FALSE);                /* return FALSE for end of file     */

    forall(i,ncom) {                  /* for each committee in network    */
        *pcl++=tmp&1;                 /* extract desired committee output */
        tmp>>=1;                      /* advance to next committee        */
    }

    *pcl=1;                   /* augment with a 1 to prevent singularity  */
    pats_so_far++;            /* update pattern sequence counter          */

    return(TRUE);             /* return TRUE if class read successfully   */
}


/***********************************************************************
 *
 *    R E A D   P A T T E R N  --  R e a d   N e x t   P a t t e r n
 *
 ***********************************************************************/

boolean read_pattern() {      /* read next pattern from training file     */

    int        i,j;           /* loop counters for row & column of image  */
    element    *pe=pattern;   /* pointer to element of pattern vector     */
    float      tmp;           /* temp cell for input conversion           */
```

```
    forall(i,patwide)                       /* for each row in the image,    */
        forall(j,pathite)                   /* for each pixel in that row,   */
            if( fscanf(pat,"%f",&tmp)       /* input value of pixel          */
                !=1 )  return(FALSE);       /* return FALSE if end-of-file   */
            else *pe++=(element)tmp;        /* convert to type element       */

    return( read_class() );      /* read in its class as an array
      * of correct decisions for each committee in the network.  If the
      * entire pattern is read, together with its class, return TRUE. */
}


/*************************************************************************
 *
 *      C O U N T   V O T E S  --  C o u n t   T h e   V o t e s
 *
 *************************************************************************/

int count_votes(pc) /* count the votes for each tlu in a committee      */
committee  *pc;     /* second parameter is a pointer to committee       */
{
    DOT *pd=pc->dot;          /* dot product save cell pointer          */
    tlu *pt=pc->wtpt;         /* tlu pointer                            */

    int     ti,               /* tlu index (loop counter)               */
            count=0;          /* the count of votes for the committee   */

    forall(ti,ntlu)                              /* forall tlus in committee */
        count+=sign(                             /* count votes as + or -    */
            *pd++=                               /* & save dot product as    */
                dotprod(*pt++,pattern)           /* weight point dotted with */
        );                                       /* pattern vector           */
    return(count);            /* return tally                           */
}


/*************************************************************************
 *
 *     R E C O G N I Z E  --  R e c o g n i z e   A   P a t t e r n
 *
 *************************************************************************/

void recognize() {  /* recognize a pattern by taking the decision
                     * of each committee to be a bit in the category
                     * number for the pattern
                     */

    int     i,          /* loop counter                                  */
            *pv=vote;   /* pointer to vote count array                   */

    boolean *pdec=decsn;/* pointer to decision array.
                         * this holds the decision bits for each
                         * of the committees in the network.
                         */

    committee *pc=net;  /* pointer to current committee in network      */

    forall(i,ncom)        /* for all committees in the network...        */
        *pdec++=alpha(*pv++=count_votes(pc++)); /* how many votes ?     */
}


/*************************************************************************
 *
 *   $ G E T   W E A K   T L U  --  S w a y   W h i c h   O n e  ?
 *
 *************************************************************************/

int get_weak_tlu(ci)    /* choose tlu most vulnerable to be swayed      */
int ci;                 /* argument is committee index                  */
{
    int     weak=0,              /* index of weakest tlu so far          */
            sv=isign(vote[ci]),  /* sign of committee's vote             */
            ti;                  /* tlu index                            */

    DOT *pd=(&net[ci])->dot,          /* pointer to dot product array    */
        conviction=INFINITY,          /* lowest conviction so far        */
        d;                            /* saved dot product value         */

    forall(ti,ntlu) {   /* for all of the tlu's in this committee...      */

        d=pd[ti];            /* get the saved dot product value          */

        if(sign(d)==sv) {               /* if tlu voted incorrectly      */
```

(continued on next page)

Listing One *(Listing continued, text begins on page 16.)*

```
            if(eabs(d)<conviction) {          /* and if this tlu has the
              * least conviction of any that have been examined so far, */

                weak=ti;    /* then remember it as the best one so
                  * far to adjust to sway the vote of this committee. */

                conviction=eabs(d);     /* update lowest conviction     */
            }
        }
    }
    return(weak);       /* return subscript of weakest tlu in committee */
}


/************************************************************************
 *
 *  A D J U S T M E N T -- C o r r e c t i o n   C o e f f i c i e n t
 *
 ************************************************************************/

element adjustment(ci,ti)   /* compute correction coefficient          */
int     ci,                 /* committee index                         */
        ti;                 /* tlu index                               */
{
    DOT d=(&net[ci])->dot[ti];          /* saved dot product            */

    if(corr_incr)                       /* fixed increment correction   */
        return(corr_incr*sign(d));

    if(absolute)                        /* absolute correction          */
        return((int)(d/patmag)+sign(d));

    if(fraction)                        /* fractional correction        */
        return(d*fraction/patmag);

    return(abort("No correction method specified."));
}


/************************************************************************
 *
 *      A D J U S T -- C h a n g e   T L U ' s   W e i g h t s
 *
 ************************************************************************/

void adjust(ci,ti)  /* adjust the weights of a single tlu              */
int ci,             /* committee index                                 */
    ti;             /* tlu index                                       */
{
    vector pw=(&net[ci])->wtpt[ti],     /* pointer to a weight          */
           pp=pattern;                  /* pointer to a pixel           */

    element lambda=adjustment(ci,ti),   /* the correction coefficient   */
            wt,awt;                     /* temps for max weight point   */

    int     i;                          /* element index & loop counter */

    tlus_trained++;                     /* count adjustment of tlu      */

    forall(i,dim) {                             /* for each coefficient */
        wt=(*pw++)-=lambda*(*pp++);             /* adjust weights       */
        awt=eabs(wt);                           /* save magnitude       */
        if(maxel<awt) {                         /* new maximum ???      */
            maxel=awt;                          /* yes, update max elem */
            if(log_level) {                     /* if any logging,      */
                printf("\nmaxel=%f",            /* then display the     */
                        (float)maxel);          /* new maximum value    */
            }
        }
    }

    if(log_level>=3)
        printf("\n        com=%d    tlu=%d    lambda=%g",
                          ci,ti,(float)lambda);
}


/************************************************************************
 *
 * S W A Y   T L U S -- S w a y   T L U s   T o   C h a n g e   V o t e
 *
 ************************************************************************/
```

```
void sway_tlus(ci)  /* sway enough tlu's to change the vote           */
int ci;             /* parameter is committee index                   */
{
    int i,                    /* loop counter                         */
        lost_by=iabs(vote[ci]/2)+1, /* how many votes we lost by      */
        weak_tlu;             /* weakest wrong tlu in committee       */

    DOT *pd=(&net[ci])->dot;  /* pointer to dot product array         */

    forall(i,lost_by) {   /* do this enough times to sway the vote... */

        weak_tlu=get_weak_tlu(ci);  /* find most vulnerable tlu       */

        adjust(ci,weak_tlu);        /* adjust its weights to change
                                     * its mind about the pattern */
        pd[weak_tlu]=-sign(pd[weak_tlu]); /* flip sign of dot product
          * so this tlu won't be considered again in this loop */
    }
}


/**********************************************************************
 *
 *      S H O W   B I T S -- D i s p l a y   B i t s   O n   C R T
 *
 **********************************************************************/

void show_bits(ps,pb)   /* display a bit vector on the screen         */
char     *ps;           /* the label for the bit vector               */
boolean *pb;            /* the pointer to the bit vector              */
{
    int i,              /* loop counter                               */
        k=1,            /* power of two                               */
        v=0;            /* value accumulator                          */

    forall(i,ncom) {        /* for all committees                     */
        if(*pb++) v+=k;     /* convert binary to decimal              */
        k<<=1;              /* advance to next bit                    */
    }
    printf("     %s %d",ps,v);     /* display label and value         */
}


/**********************************************************************
 *
 *      T R A I N -- T r a i n   T h e   N e t w o r k
 *
 **********************************************************************/

train() {        /* train the network to recognize the pattern        */

    int     ci;     /* committee index                                */

    goofed=FALSE;       /* give benefit of doubt -- assume didn't goof */

    patmag=dotprod(pattern,pattern);    /* find pattern magnitude     */

    forall(ci,ncom)     /* for all the committees in the network...   */

        if(decsn[ci]!=class[ci]) {      /* if the committee goofed up, */
            goofed=TRUE;                /* then say so,               */
            pats_missed++;              /* count misrecognized pattern, */
            sway_tlus(ci);              /* and change enough tlu's
               * so it won't goof up on this pattern next time ! */
        }

    if(goofed) {                        /* did we goof?               */
        missed++;                       /* yes, count the boo boo!    */
        if(log_level>=2) {              /* if detail requested,       */
            printf("\n");               /* start a new line           */
            show_bits("siloam ",decsn); /* show machine's decision    */
            show_bits("really ",class); /* display what really is     */
        }
    }
}

/**********************************************************************
 *
 *   T O T C O N S -- T o t a l   N u m b e r   O f   C o n n e c t s
 *
 **********************************************************************/

int totcons() {                       /* count total # of connections */
```

(continued on next page)

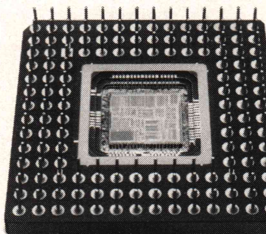
Listing One *(Listing continued, text begins on page 16.)*

```
    committee  *n=net;                    /* neural network pointer        */
    tlu        *c,                        /* committee pointer             */
               t;                         /* tlu pointer                   */
    int        i,j,k,                     /* loop indices                  */
               no=0;                      /* totalizer accumulator         */

    forall(i,ncom) { c=n++->wtpt;         /* for each committee...         */
        forall(j,ntlu) { t=*c++;          /* for each tlu in the committee*/
            forall(k,dim-1)               /* for each element in the tlu   */
                if(*t++!=0) no++;         /* count it if it is connected   */
        }
    }
    return(no);                           /* return the count              */
}

/**********************************************************************
 *
 *   S I L O A M    O u t s i d e    C o n t r o l    S t r u c t u r e
 *
 **********************************************************************/

void siloam() {      /* outside control structure for pattern recognizer */

    long start,stop;    /* timer value cells for benchmarking             */
    int  cons,new,old=0;/* connection counters                            */

    read_header();      /* read header information in the training file */

    initialize();       /* allocate the committees of TLUs and
                         * initialize the weight points randomly */

    radius_statistics();/* print starting radius statistics              */

    npass=0;            /* initialize pass counter                        */
    start=time(NULL);   /* remember start time                            */

    do {                /* start over in training file,
                         * we made a mistake... */

        missed=0;       /* reset misrecognition counter                   */

        read_header();      /* rewind training file
                             * and skip over header information... */

        while(read_pattern()) { /* keep reading patterns until we've
          * done the entire training file and recognized them all */

            recognize();        /* attempt to recognize the pattern     */

            train();            /* adjust any weights necessary to get
                                 * the correct recognition if we goofed */

            if(goofed&&start_over) break;   /* select training strategy */

        }                   /* end of while loop to read next pattern   */

        npass++;                          /* increment pass counter        */
        if(log_level>=1) {                /* give pass summary report      */
            cons=totcons();               /* count the connections         */
            new=cons-old;                 /* compute how many new ones     */
            old=cons;                     /* remember for next time        */
            printf("\npass # %d    missed %d    cons=%d    new=%d",
                       npass,        missed,      cons,      new);
        }
    } while(missed);        /* end of do loop to train network            */

    stop=time(NULL);        /* get stop time                              */

    /***************** print end of run summary *************************/

    printf("\n");
    printf("\ntraining completed in %ld seconds.\n",stop-start);
    printf("\nnumber of committees:   %d",ncom            );
    printf("\nnumber of tlus total:   %d",ncom*ntlu       );
    printf("\nnumber of elements:     %d",ncom*ntlu*dim   );
    printf("\nnumber of connections: %d",totcons()       );
    printf("\n");

    printf("\nnumber of passes thru file: %d",npass);
    printf("\nnumber of patterns in file: %d",pats_so_far );
    printf("\nnumber of mis-recognitions: %d",pats_missed );
    printf("\nnumber of tlu adjustments:  %d",tlus_trained);
    printf("\nmaximum element magnitude:  %f",(float)maxel);
```

```
    printf("\n");

    radius_statistics();    /* print ending radius statistics           */

}


/************************************************************************
 *
 *       M A I N   P r o g r a m   S t a r t s   H e r e
 *
 ************************************************************************/

main(paramct,params)      /********** main program entry point ************/

int  paramct;             /* number of parameters on command line          */
char *params[];           /* array of pointers to strings for each param   */

{
    int i;                /* array index variable                          */

    banner();             /* print program name, version, & release date   */

    printf("\nInvoked By:");                    /* show how the program */
    for(i=1;i<=paramct;i++) printf(" %s",params[i]); /* was started up! */
    printf("\nelement type is %s",eltype);      /* show arithmetic used */
    printf("\n");

    /********************** parse the command line *********************/

    if(paramct==1) help();  /* if no params, then give help and quit !  */
    patname[0]=0;           /* else set pattern filename to null string */

    for (i=1;i<paramct;i++) {           /* for each parameter...        */

        if('-'==params[i][0])               /* is it an option ?        */

            switch(toupper(params[i][1])) { /* yes, which one ?         */

                kase('O',start_over=TRUE)                  /* strategy     */
                kase('L',log_level=atoi(&params[i][2]))  /* log detail   */
                kase('T',ntlu=atoi(&params[i][2]))       /* # of TLUs    */
                kase('R',radius=atof(&params[i][2]))     /* init radius  */
                kase('I',corr_incr=atoi(&params[i][2]))  /* fixed incr   */
                kase('A',absolute=TRUE)                  /* absolute     */
                kase('F',fraction=atof(&params[i][2]))   /* fractional   */
            }

        /****************** parse filename ***************************/

        else if(index(&params[i][0],'.'))   /* is '.' in it?            */
            move(&params[i][0],patname);    /* yes, pattern file        */

        else move(".PAT",                   /* no, default extension is */
            move(&params[i][0],patname));   /* '.pat' for pattern file  */
    }

    /**************** check for command line errors *********************/

    if(patname[0]==0)       /* check for missing pattern file name      */
        abort(
        "pattern filename not specified!");

    if(ntlu==0)             /* check for missing number of TLUs         */
        abort(
        "number of TLUs per committee not specified!");

    /************************ open pattern file ***********************/

    if(!(pat=fopen(patname,"r")))           /* if open fails, abort     */
        abort(
        "can't open pattern file!");

    /********* perform the training and recognition algorithm ***********/

/*  srand(1);  */    /* make random number generator repeatable --
                      * ...this may be removed, if desired, after the
                      * debug phase is complete! */

    siloam();        /* call the outside control structure for the
                      * trainable pattern recognizer. */
}
```

End Listing

A TRAINING COURSE FOR PEOPLE WHO LUST AFTER POWER.



The 80386

Intel is offering three new courses on the world's most powerful 32-bit microprocessor – the 80386. Plus a new intensive course on the 80286.

These in-depth learning sessions are designed for engineers and programmers who want to utilize the full power and potential of these lightning-fast chips.

Lectures are combined with hands-on workshops to provide real-life situations. Allowing you to apply new concepts and techniques immediately.

Courses include:

80386 System Software
80386 Programming using ASM386
High-end Microprocessor Hardware Design
The new 80286 Microprocessor Family Course

Complete training sessions and courses can be scheduled at your facility, or at our training centers.

In addition to training, Intel offers hardware/software support and consultants.

For more complete course information and schedules, call toll-free (800) 548-4725.

Or to register now, contact one of the Intel Training Centers listed below.

Intel Training Centers

Boston Area, Westford Corp. Ctr.
Three Carlisle Road, 1st Floor
Westford, MA 01886
(617) 692-1000

Chicago Area
300 N. Martingale Road, Suite 300
Schaumburg, IL 60194
(312) 310-5700

San Francisco Area
2700 San Tomas Expressway
Santa Clara, CA 95051
(408) 970-1700

Washington, D.C. Area
7833 Walker Drive, 5th Floor
Greenbelt, MD 20770
(301) 220-3380

intel®

Circle **no. 190** on reader service card.

Listing One *(Text begins on page 42.)*

```
REM Mycin-like expert system for Amiga BASIC
REM by Richard Grigonis

DIM AND.COMPONENT(12),AT.FACTOR.FOR.OR.COMPONENT(12)
DIM OR.COMPONENT(12),TRAIL(30),HUMAN.INPUT$(4)
DIM MESSAGE$(60),WHICH.EQ$(8),BLANK$(19)
DIM HYPOTHESIS(20) ' CHANGE THIS NUMBER IF MORE THAN 20 ANIMALS

no=0:yes=1
DATA 1,is an albatross
albatross=1
DATA 2,is a penguin
penguin=2
DATA 3,is an ostrich
ostrich=3
DATA 4,is a zebra
zebra=4
DATA 5,is a giraffe
giraffe=5
DATA 6,is a tiger
tiger=6
DATA 7,is a cheetah
cheetah=7
DATA 8,flies well
flies.well=8
DATA 9,swims
swims=9
DATA 10,is black and white
black.and.white=10
DATA 11,cannot fly
cannot.fly=11
DATA 12,has a long neck
long.neck=12
DATA 13,has black stripes
black.stripes=13
DATA 14,has long legs
long.legs=14
DATA 15,has dark spots
dark.spots=15
DATA 16,has a tawny color
tawny.color=16
DATA 17,is a bird
bird=17
DATA 18,is an ungulate
ungulate=18
DATA 19,is a carnivore
carnivore=19
DATA 20,is a mammal
mammal=20
DATA 21,has hair
has.hair=21
DATA 22,gives milk
gives.milk=22
DATA 23,eats meat
eats.meat=23
DATA 24,has pointed teeth and claws and forward pointing eyes
teeth.claws.eyes=24
DATA 25,is a mammal and has hoofs
mammal.and.hoofs=25
DATA 26,is a mammal and chews cud
mammal.and.chews.cud=26
DATA 27,has feathers
feathers=27
DATA 28,flies and lays eggs
flies.and.lays.eggs=28
DATA 29,lays eggs
lays.eggs=29
DATA 30,flies
flies=30
DATA 31,chews cud
chews.cud=31
DATA 32,has hoofs
hoofs=32
DATA 33,has forward pointing eyes
front.eyes=33
DATA 34,has claws
claws=34
DATA 35,has pointed teeth
pointed.teeth=35
DATA -1,END OF DATA

REM TOP-LEVEL HYPOTHESES (ROOTS) OF AND/OR TREE:
HYPOTHESIS(1)=albatross
HYPOTHESIS(2)=penguin
HYPOTHESIS(3)=ostrich
HYPOTHESIS(4)=zebra
HYPOTHESIS(5)=giraffe
HYPOTHESIS(6)=tiger
HYPOTHESIS(7)=cheetah
number.of.hypotheses=7

REM DETERMINE TOTAL NUMBER OF FACTS:
number.of.facts=0
WHILE fact <> -1
  READ fact,MESSAGE$
  number.of.facts=number.of.facts+1
WEND
number.of.facts=number.of.facts-1
DIM BEEN.EXAMINED.BEFORE(number.of.facts),OUTPUT.CF
                                                  (number.of.facts)

Start:
  FOR A=0 TO UBOUND(OUTPUT.CF)
    OUTPUT.CF(A)=0:BEEN.EXAMINED.BEFORE(A)=0
  NEXT A
  PRINT "I'm a backward-chaining expert system."
  PRINT "Please think of one of the";number.of.hypotheses
  PRINT "animals listed below.  I will ask you"
  PRINT "questions about the animal and compute"
```

```
  PRINT "the certainty of it being one"
  PRINT "of the following";number.of.hypotheses;"animals:":PRINT
  FOR fact=1 TO number.of.hypotheses
    which.fact=HYPOTHESIS(fact)
    GOSUB Find.message:PRINT "ANIMAL ";MESSAGE$
  NEXT fact
  PRINT

10030 PRINT "DO YOU WANT: "
PRINT "AN EXHAUSTIVE SEARCH (1) OR,"
PRINT "STOP-ON-SUCCESS (2)? ":PRINT
PRINT "Press the NUMBER of YOUR SELECTION"
PRINT "and then press the RETURN KEY."
halt.on.success=0:INPUT halt.on.success
IF 0>halt.on.success OR halt.on.success>2
   THEN PRINT "TRY AGAIN!":GOTO 10030

10050 REM PROVE HYPOTHESES
GOSUB Prove.albatross
IF halt.on.success=2 AND OUTPUT.CF(albatross)=1 THEN 10165
GOSUB Prove.penguin
IF halt.on.success=2 AND OUTPUT.CF(penguin)=1 THEN 10165
GOSUB Prove.ostrich
IF halt.on.success=2 AND OUTPUT.CF(ostrich)=1 THEN 10165
GOSUB Prove.zebra
IF halt.on.success=2 AND OUTPUT.CF(zebra)=1 THEN 10165
GOSUB Prove.giraffe
IF halt.on.success=2 AND OUTPUT.CF(giraffe)=1 THEN 10165
GOSUB Prove.tiger
IF halt.on.success=2 AND OUTPUT.CF(tiger)=1 THEN 10165
GOSUB Prove.cheetah
IF halt.on.success=2 AND OUTPUT.CF(cheetah)=1 THEN 10165

10165 REM DISPLAY RESULTS
CLS
PRINT "HERE ARE THE COMPUTED CERTAINTY FACTORS:"
PRINT "(Correct animal has highest positive CF#)":PRINT
FOR fact=1 TO number.of.hypotheses
  which.fact=HYPOTHESIS(fact)
  GOSUB Find.message
  BLANK$=SPACE$(19)
  MESSAGE$=MESSAGE$+MID$(BLANK$,1,LEN(BLANK$)-LEN(MESSAGE$))
  PRINT "ANIMAL ";MESSAGE$;" CF=";OUTPUT.CF(which.fact)
NEXT fact
PRINT:PRINT "TO GO AGAIN, press the RETURN button."
INPUT HUMAN.INPUT$:PRINT
GOTO Start

REM SUBROUTINES TO COMPUTE CF'S (IN ALPHABETICAL ORDER)
Compute.and.clause.cf:
  GOSUB Find.lowest.cf.branch
  GOSUB Multiply.lowest.cf.by.at.factor
  GOSUB Trim.to.zero
  OUTPUT.CF(TRAIL(depth))=new.cf
RETURN

Compute.or.clause.cf:
  GOSUB Multiply.component.cfs.by.at.factors
  GOSUB Test.for.a.positive.number
  GOSUB Run.or.equation
  GOSUB Trim.to.zero
  OUTPUT.CF(TRAIL(depth))=new.cf
RETURN

Dec.stack:
  depth=depth-1:RETURN
Deduce:
  which.fact=TRAIL(depth):GOSUB Find.message
  PRINT:PRINT "The fact that the animal "
  PRINT MESSAGE$;" (FACT # ";fact.number;")"
  PRINT "Now has a Certainty Factor of: ";OUTPUT.CF(TRAIL(depth))
  PRINT:GOSUB Dec.stack:GOSUB Delay
RETURN

Delay:
  FOR D=1 TO 10000:NEXT D:RETURN
Explain.why:
  which.fact=TRAIL(1):GOSUB Find.message:CLS
  PRINT "I AM INVESTIGATING THE HYPOTHESIS"
  PRINT "THAT THE ANIMAL..."
  PRINT MESSAGE$;" (FACT # ";fact.number;")":PRINT
  IF depth=1 THEN
    PRINT "...BY FIRST ASKING YOU.":PRINT
    PRINT "If you are not sure (-8 < CF < 8)"
    PRINT "then I will investigate this hypothesis further."
  ELSE
    FOR A=2 TO depth
      which.fact=TRAIL(A)
      GOSUB Find.message:PRINT "...BY PROVING THAT THE ANIMAL..."
      PRINT MESSAGE$;" (FACT # ";fact.number;")":PRINT
    NEXT A
    PRINT "...BY ASKING YOU."
  END IF
  PRINT:INPUT "PRESS RETURN KEY TO CONTINUE",HUMAN.INPUT$
RETURN

Find.lowest.cf.branch:
  lowest.number=OUTPUT.CF(AND.COMPONENT(1))
  FOR branch=1 TO number.of.and.clause.components
    number.to.test=OUTPUT.CF(AND.COMPONENT(branch))
    IF lowest.number>number.to.test
      THEN lowest.number=number.to.test
  NEXT branch
RETURN

Find.message:
  RESTORE
  FOR C=1 TO which.fact
```

(continued on next page)

Listing One

(Listing continued, text begins on page 42.)

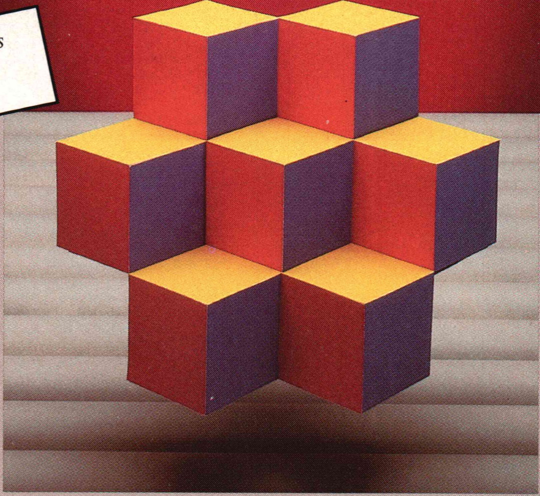
```
    READ fact.number, MESSAGE$
  NEXT C
RETURN

Inc.stack:
  depth=depth+1:TRAIL(depth)=current.fact
RETURN

Multiply.component.cfs.by.at.factors:
  FOR branch=1 TO number.of.or.clause.components
    new.cf=OUTPUT.CF(OR.COMPONENT(branch))
     *AT.FACTOR.FOR.OR.COMPONENT(branch)
    GOSUB Trim.to.zero
    OUTPUT.CF(OR.COMPONENT(branch))=new.cf
  NEXT branch
RETURN

Multiply.lowest.cf.by.at.factor:
  new.cf=lowest.number*at.factor.for.and.clause
RETURN

Negative.or.equation:
  new.cf=1
  FOR branch=1 TO number.of.or.clause.components
    new.cf=new.cf*(1+OUTPUT.CF(OR.COMPONENT(branch)))
  NEXT branch
  new.cf=-1+new.cf
RETURN

Positive.or.equation:
  new.cf=1
  FOR branch=1 TO number.of.or.clause.components
    new.cf=new.cf*(1-OUTPUT.CF(OR.COMPONENT(branch)))
  NEXT branch
  new.cf=1-new.cf
RETURN

Run.or.equation:
  IF WHICH.EQ$="POSITIVE" THEN
    GOSUB Positive.or.equation
  ELSE
    GOSUB Negative.or.equation
  END IF
RETURN

Test.fact.for.human.input:
  leave=no:GOSUB Inc.stack
  IF BEEN.EXAMINED.BEFORE(current.fact)=yes THEN leave=yes:GOSUB
Dec.stack:RETURN
  BEEN.EXAMINED.BEFORE(current.fact)=yes
4156 which.fact=current.fact
    GOSUB Find.message
4160 CLS:PRINT"(FACT # ";fact.number;")":PRINT
    PRINT "ON A SCALE OF -10 TO 10 WHERE,"
    PRINT " 10=absolutely certain it's true"
    PRINT "  8=almost certain"
    PRINT "  6=probably"
    PRINT "  3=slight evidence"
    PRINT "  0=unknown"
    PRINT " -6=probably not"
    PRINT " -8=almost certainly not"
    PRINT "-10=definitely not"
    PRINT:PRINT"TO WHAT DEGREE DO YOU BELIEVE THAT"
    PRINT "The animal ";MESSAGE$;"?":PRINT
    PRINT "TYPE NUMBER AND PRESS RETURN KEY,"
    PRINT "OR TYPE `why?' AND PRESS RETURN KEY"
    INPUT HUMAN.INPUT$
    HUMAN.INPUT$=UCASE$(HUMAN.INPUT$)
    IF HUMAN.INPUT$="WHY" OR HUMAN.INPUT$="WHY?"
      THEN GOSUB Explain.why:GOTO 4156
    I=VAL(HUMAN.INPUT$)
    IF -10>I OR I>10 THEN GOTO 4160
    I=I/10:OUTPUT.CF(current.fact)=I
    IF -.8>I OR I>.8 THEN leave=yes:GOSUB Deduce
RETURN

Test.for.a.positive.number:
  WHICH.EQ$="NEGATIVE"
  FOR branch=1 TO number.of.or.clause.components
    number.to.test=OUTPUT.CF(OR.COMPONENT(branch))
    IF number.to.test>0 THEN
WHICH.EQ$="POSITIVE":branch=number.of.or.clause.components
  NEXT branch
RETURN

Trim.to.zero:
  IF -.2<=new.cf AND new.cf<=.2 THEN
    new.cf=0
  ELSEIF new.cf>=.8 THEN
    new.cf=1
  ELSEIF new.cf<=-.8 THEN
    new.cf=-1
  END IF
RETURN

REM ***DEDUCTIVE ROUTINES FOLLOW***
Prove.albatross:
  current.fact=albatross:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.bird:GOSUB Prove.flies.well
    number.of.and.clause.components=2
      AND.COMPONENT(1)=bird:AND.COMPONENT(2)=flies.well
    at.factor.for.and.clause=1
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN

Prove.penguin:
```

(continued on page 78)

WHY LOGITECH MODULA-2 IS MORE POWERFUL THAN PASCAL OR C.

"A clear winner…The integrated editor is a joy to use." BYTE Magazine, Jan. '87

APPRENTICE PACKAGE $99



WIZARDS' PACKAGE $199

- Separate Compilation w/inter-module typechecking
- Native Code Generation
- Large Memory Model Support
- Most Powerful Runtime Debugger
- Comprehensive Module Library
- Maintainability
- Translator from Turbo and ANSI Pascal

WIN A FREE TRIP TO Switzerland

HOMELAND OF MODULA-2

Return your Modula-2 Registration Card or a reasonable facsimile* postmarked between March 1, 1987 and May 31, 1987 to be included in a once-only drawing!

Grand Prize: One week excursion for 2 in Zurich, Switzerland including a guided tour of ETH, the University where Modula-2 was created by Niklaus Wirth. European customers may substitute a trip to Silicon Valley, California.

Second and Third Prizes: LOGITECH C7 Mouse or LOGITECH Bus Mouse with Paint & Draw software—a $219 value, absolutely free!

*Write to Logitech, Inc. for a registration card facsimile.

NEW!

APPRENTICE PACKAGE $99

Everything you need to begin producing reliable maintainable Modula-2 code. Includes the Compiler with 8087 support, integrated Editor, Linker, and BCD Module. We're also including FREE our Turbo Pascal to Modula-2 Translator!

NEW!

WIZARDS' PACKAGE $199

This package contains our Plus Compiler—for professional programmers or for those who just want the best. The Plus Compiler with Integrated Editor requires 512K and takes advantage of the larger memory to increase compilation speed by 50%. Our Turbo Pascal to Modula-2 Translator is also included at no extra charge.

NEW!

MAGIC TOOLKIT $99

We've put our most powerful development tools into one amazing Toolkit for use with either the Apprentice or Wizards' packages. Highlighted by our Runtime Debugger, the finest debugging tool available anywhere, the Toolkit also includes our Post Mortem Debugger, Disassembler, Cross Reference utility and Version which keeps track of different versions of one program. Our MAKE Utility figures out module dependencies and automatically selects those affected by code changes to minimize recompilation and relinking. We also provide source code of our major library modules for you to customize—or just play with.

WINDOW PACKAGE $49

Now you can build true windowing into your Modula-2 code. Features virtual screens, color support, overlapping windows and a variety of borders.

ROM PACKAGE AND CROSS RUN TIME DEBUGGER $299

For those who want to produce rommable code. You can even debug code running in ROM from your PC.

Turbo Pascal is a registered trademark of Borland International.

Call for information about our VAX/VMS version, Site License, University Discounts, Dealer & Distributor pricing.

To place an order call toll-free:

800-231-7717

In California:

800-552-8885

YES! I want the spellbinding power of LOGITECH Modula-2!

☐ Apprentice Package	$99
☐ Wizards' Package	$199
☐ Magic Toolkit	$99
☐ Window Package	$49
☐ ROM Pkg/Cross RTD	$299

Add $6.50 for shipping and handling. Calif. residents add applicable sales tax. Prices valid in U.S. only.

Total Enclosed $______

☐ VISA ☐ MasterCard ☐ Check Enclosed

Card Number ______ Expiration Date

Signature ______

Name ______

Address ______

City ______ State

Zip ______ Phone

LOGITECH

LOGITECH, Inc.
805 Veterans Blvd. Redwood City, CA 94063
Tel: 415-365-9852

In Europe:
LOGITECH SA, Switzerland
Tel: 41-21-879656 • Telex 458 217 Tech Ch

In Italy:
Tel: 39-2-215-5622

Circle **no. 257** on reader service card.

Listing One

(Listing continued, text begins on page 42.)

```
  current.fact=penguin:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.bird:GOSUB Prove.cannot.fly
  GOSUB Prove.black.and.white:GOSUB Prove.swims
    number.of.and.clause.components=4
      AND.COMPONENT(1)=bird:AND.COMPONENT(2)=cannot.fly
      AND.COMPONENT(3)=black.and.white:AND.COMPONENT(4)=swims
    at.factor.for.and.clause=.8
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN

Prove.ostrich:
  current.fact=ostrich:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.bird:GOSUB Prove.cannot.fly
  GOSUB Prove.black.and.white:GOSUB Prove.long.neck
    number.of.and.clause.components=4
      AND.COMPONENT(1)=bird:AND.COMPONENT(2)=cannot.fly
      AND.COMPONENT(3)=black.and.white:AND.COMPONENT(4)=long.neck
    at.factor.for.and.clause=.85
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN

Prove.zebra:
  current.fact=zebra:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.ungulate:GOSUB Prove.black.stripes
    number.of.and.clause.components=2
      AND.COMPONENT(1)=ungulate:AND.COMPONENT(2)=black.stripes
    at.factor.for.and.clause=.8
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN

Prove.giraffe:
  current.fact=giraffe:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.ungulate:GOSUB Prove.long.neck
  GOSUB Prove.long.legs:GOSUB Prove.dark.spots
    number.of.and.clause.components=4
      AND.COMPONENT(1)=ungulate:AND.COMPONENT(2)=long.neck
      AND.COMPONENT(3)=long.legs:AND.COMPONENT(4)=dark.spots
    at.factor.for.and.clause=.85
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN

Prove.tiger:
  current.fact=tiger:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.mammal:GOSUB Prove.carnivore
  GOSUB Prove.black.stripes:GOSUB Prove.tawny.color
    number.of.and.clause.components=4
      AND.COMPONENT(1)=mammal:AND.COMPONENT(2)=carnivore
      AND.COMPONENT(3)=black.stripes:AND.COMPONENT(4)=tawny.color
    at.factor.for.and.clause=.95
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN

Prove.cheetah:
  current.fact=cheetah:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.mammal:GOSUB Prove.carnivore
  GOSUB Prove.tawny.color:GOSUB Prove.dark.spots
    number.of.and.clause.components=4
      AND.COMPONENT(1)=mammal:AND.COMPONENT(2)=carnivore
      AND.COMPONENT(3)=tawny.color:AND.COMPONENT(4)=dark.spots
    at.factor.for.and.clause=.95
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN

Prove.flies.well:
  current.fact=flies.well:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN
Prove.swims:
  current.fact=swims:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.black.and.white:
  current.fact=black.and.white:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.cannot.fly:
  current.fact=cannot.fly:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.long.neck:
  current.fact=long.neck:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.black.stripes:
  current.fact=black.stripes:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
```

(continued on page 82)

WE'RE LOOKING FOR A LASTING RELATIONSHIP.

When we founded Programmer's Connection in 1984, we dedicated ourselves to providing our customers with personal service, low prices and the highest level of support possible. Our goal was to make *every* customer a lifelong customer.

And we're close to our goal. According to the results of our customer service questionnaire, 99% of those responding said they would buy from us again.

Last October, we began including our questionnaire in every outgoing package. It asks our customers to rate each area of service and invites them to tell us what we need to improve as well as what we're doing right.

Since then, we received almost a thousand responses with many valuable comments and suggestions. And we're quite pleased that so many of them were highly favorable.

So call Programmer's Connection today and find the lasting relationship you've always been looking for. You'll be glad you did!

Here are some comments offered by people responding to our customer service questionnaire:

"If only all the companies I used were as good as Programmer's Connection!"

"The best service for professional users in the country."

"As a dealer specializing in programming, I've found that you are my best source."

"Thank you for all your courteous assistance."

"Programmer's Connection is really outstanding. Good prices, prompt shipment, no extra fees. Wonderful 30day trial."

"There is a lot of competition in your field, but Programmer's Connection is the best!"

"It's a pleasure to deal with a company that understands the needs of the computer professional. Thanks."

"You're the best in the business! Keep it up!"

"Your service and prices are outstanding. I'm glad I found you."

"Keep up the good work. I like the way you stay up to date."

"I appreciate your courteous, reliable service."

"This method of purchasing software allows good selection and convenience."

"Good products and great prices!"

"It's unusual to find a company that offers the best price and the best service. Programmer's Connection does — keep up the good work!"

"You have the best prices."

"I particularly like your pricing policy — lower than anyone else and no extras for shipping, credit cards, etc. Keep up the good work!"

"With the service I have received from you, I am sure to look at Programmer's Connection next time I buy."

"I appreciate your professional attitude. It's refreshing."

"Class "A" performance!"

"Very pleasant, helpful order taker. I'm impressed."

"You are the FIRST place I go for professional software products. Thanks! (Especially for your 30-day trial service)."

"Product was out of stock, but shipped exactly as I was told it would be. Good Job!"

"You provide a great service to me as a professional programmer. You are #1 on my list. Keep up the excellent work!"

"Your company is a pleasure to do business with."

"Thanks for providing prompt, reliable service and delivery."

"I'm extremely happy — just placed my 6th order."

"Anything you carry, I buy from you. You can teach your competition a lesson!"

"You have great service! You have the best prices!"

"An oasis in the mail-order desert. Keep up the excellent work!"

"Amazing — no problems — I'll be back."

"Probably the best service of any well advertised company."

"I always recommend you when asked where to buy software."

"I have not found any mail order company with such competitive prices whose people are as helpful and knowledgeable as yours. I truly appreciate it and will order more from you. Keep up the good job."

"You have both lowest prices and best service — a great combination! Keep up the good work."

"You guys are SUPER."

"Technical staff very helpful."

"Once again, it was a pleasure dealing with you."

"I'm extremely satisfied with all aspects of your operation with which I am familiar."

"Excellent service. I most definitely will be purchasing from you in the near future. Keep up the good work."

"Impressive service and pricing. I'll recommend you to my friends."

"Your prices are the best! It's great that you pick up the shipping charges and charge no sales tax (for me anyway). The price I see advertised is the price I pay, period."

"GOOD JOB!"

"I referred you to two others because they could not get current versions from other vendors."

"Your service is unusual for purchasing software."

"The person who took my phone order seemed to be genuinely eager to help me, not just get an order. Thanks."

"I like having no surcharge on credit cards and no shipping charge. There is no hidden cost."

"Good products + good service + good prices = happy customers. Keep it up."

"Fastest I've ever received by mail order. Good job. You have my patronage."

"Good service at very good prices. I plan to make all purchases from you."

"You sell up-to-date products at reasonable prices and good support."

PROGRAMMER'S CONNECTION — QUALITY · INTEGRITY · Service · SUPPORT · PRICE

Turn the page for our latest advertised price list.

ai - arity products

Arity Combination Package	1095	979
Expert System Development Pkg	295	229
File Interchange Toolkit	50	44
PROLOG Compiler & Interpreter	650	569
Screen Design Toolkit	50	44
SQL Development Package	295	229
Arity PROLOG Interpreter	295	229
Arity Standard Prolog	95	77

ai - expert systems

1st-CLASS *by Programs in Motion*	495	399
AutoIntelligence *by IntelligenceWare*	990	749
ExpertEDGE Advanced *by Human Edge*	2500	CALL
ExpertEDGE Professional *by Human Edge*	5000	CALL
Experteach II *by IntelligenceWare*	475	349
EXSYS *Development Software by EXSYS*	395	309
EXSYS Runtime System	600	469
Insight 1 *by Level Five Research*	95	75
Insight 2+ *by Level Five Research*	485	379
Intelligence/Compiler *IntelligenceWare*	990	749
Logic-Line Series 1 *by Thunderstone*	90	85
Logic-Line Series 2 *by Thunderstone*	125	115
Logic-Line Series 3 *by Thunderstone*	150	139

ai - lisp language

GCLISP *Golden Common LISP by Gold Hill*	495	CALL
GCLISP 286 Developer *by Gold Hill*	1190	CALL
IQCLISP *by Integral Quality* New	300	CALL
IQLISP *by Integral Quality* New	270	CALL
Microsoft LISP *Common LISP*	250	149
QNIAL *Combines LISP & APL by NIAL Systems*	375	349
TransLISP *from Solution Systems*	95	CALL
TransLISP PLUS *from Solution Systems* New	195	CALL

ai - prolog language

APT *Active Prolog Tutor from Solution Systems*	65	CALL
LPA microPROLOG *All Varieties*	CALL	CALL
MPROLOG Language Primer *LOGICWARE*	50	45
MPROLOG P500 *by LOGICWARE*	495	395
MPROLOG P550 *by LOGICWARE*	220	175
Prolog-86 *from Solution Systems*	125	CALL
Prolog-86 Plus *from Solution Systems*	250	CALL
Turbo PROLOG *by Borland Intl* New Version	100	63
Turbo PROLOG Toolbox *by Borland Intl* New	100	64

ai - smalltalk language

Smalltalk/V *by Digitalk*	99	84
EGA Color Option New	49	45
Goodies Diskette New	49	45
Smalltalk/Comm	49	42

ai - texas instruments

PC Scheme *Lisp*	95	84
Personal Consultant Easy	495	435
Personal Consultant Plus	2950	2589
Personal Consultant Runtime	95	84

apl language

APL*PLUS/PC *by STSC*	595	429
APL*PLUS/PC Spreadsheet Mgr *by STSC*	195	139
APL*PLUS/PC Tools Vol 1 *by STSC*	295	199
APL*PLUS/PC Tools Vol 2 *by STSC*	85	58
Financial/Statistical Library *by STSC*	275	189
Pocket APL *by STSC*	95	69
STATGRAPHICS *by STSC*	795	579

assembly language

386 ASM/LINK *Cross Asm by Phar Lap*	495	389
8088 Assembler *w/Z-80 Translator by 2500 AD*	100	89
ASMLIB *Function Library by BC Assoc*	149	129
asmTREE *B-Tree Dev System by BC Assoc*	395	339
Cross Assemblers *Various by 2500 AD*	CALL	CALL
Microsoft Macro Assembler	150	93
Norton Utilities *by Peter Norton*	100	59
screenplay *by Flexus*	100	79
Turbo EDITASM *by Speedware*	99	84
Uniware Cross Assemblers *Various by SDS*	295	249
Visible Computer: 8088 *Software Masters*	80	64

basic language

87 QB Pak *by Hauppauge*	69	59
87 Software Pak *by Hauppauge* New	180	149
BetterBASIC *by Summit* Final Clearance	200	119
EXIM Services Toolkit *by EXIM*	50	45
Finally *by Komputerwerks*	99	85
Inside Track *from Micro Help*	65	51
MACH 2 *by Micro Help*	75	58
MACH 2 *for Turbo BASIC by Micro Help* New	69	CALL
Microsoft QuickBASIC Compiler	99	63
Peeks 'n Pokes *from MicroHelp*	45	37
Professional BASIC *by Morgan*	99	68
8087 Math Support	50	42
QuickPak *by Crescent Software* New	69	59
Scientific Subroutine Library *by Peerless*	125	99
Stay-Res *by MicroHelp*	95	73
True Basic *w/Run-time* New Version	245	179
True Basic New Version	150	97
Run-time Module	150	97
Various Utilities	50	41
Turbo BASIC Compiler *by Borland Intl* New	100	64

blaise products

ASYNCH MANAGER *Specify C or Pascal*	175	119
C TOOLS PLUS	175	119
EXEC *Program Chainer*	95	73
LIGHT TOOLS *for Datalight C*	100	78
PASCAL TOOLS	125	94
PASCAL TOOLS 2	100	74
PASCAL TOOLS & TOOLS 2	175	119
RUNOFF *Text Formatter*	50	43
TURBO ASYNCH PLUS	100	78
TURBO POWER TOOLS PLUS	100	78
VIEW MANAGER *Specify C or Pascal*	275	179

borland products

EUREKA *Equation Solver*	100	64
Reflex & Reflex Workshop	200	128
Reflex *Data Base System*	150	89
Reflex Workshop	70	45
Sidekick & Traveling Sidekick	125	85
Sidekick	85	57
Traveling Sidekick	70	45
Superkey	70	45
Turbo BASIC Compiler New	100	64
Turbo C Compiler New	100	64
Turbo Database Toolbox	70	41
Turbo Editor Toolbox	70	41
Turbo Gameworks Toolbox	70	41
Turbo Graphix Toolbox	70	41
Turbo Lightning	100	64
Turbo PASCAL Numerical Methods Toolbox	100	64
Turbo PASCAL and Tutor	125	85
Turbo PASCAL	100	64
Turbo Tutor	40	24
Turbo PROLOG Compiler New Version	100	63
Turbo PROLOG Toolbox New	100	64
Word Wizard	70	47
Word Wizard and Turbo Lightning	150	94

c++

C++ *by Guidelines w/version 1.1 kernel*	195	172
PforCe++ *Function Library by Phoenix* New	395	225

c compilers

68000/10/20 Cross Compiler *by SDS*	595	CALL
C86PLUS *by Computer Innovations*	497	CALL
Datalight C Compiler *Small Model*	60	43
Datalight Developer Kit	99	74
Datalight Optimum-C	139	109
DeSmet C *w/Debugger & Large Case*	209	184
DeSmet C *w/Debugger Only*	159	138
Eco-C *Development System by Ecosoft*	125	83
Lattice C Compiler *from Lattice*	500	269
Mark Williams Let's C Combo Pack	125	99
Let's C *Compiler*	75	54
csd *Source Level Debugger*	75	54
Mark Williams MWC-86	495	286
Microsoft C *with CodeView*	450	269
Turbo C Compiler *by Borland Intl* New	100	64
Wizard C Combo *by Wizard Systems*	750	529
Wizard C Compiler	450	299
ROM Development Pkg	350	259

c interpreters

C-terp *by Gimpel, Specify compiler*	300	219
C Trainer *with Book by Catalytix*	122	87
Instant C *by Rational Systems*	500	369
Introducing C *by Computer Innovations*	125	CALL
Run/C *by Age of Reason*	150	88
Run/C Professional *by Age of Reason*	250	157

c utilities

C to dBase *by Computer Innovations*	150	CALL
c-tree & r-tree Combo *by FairCom*	650	519
c-tree *ISAM File Manager*	395	315
r-tree *Report Generator*	295	239
C Windows *by Syscom*	100	85
C Wings *by Syscom*	50	43
CI ROMPac *by Computer Innovations*	195	CALL
dBx *dBase to C Translator by Desktop AI*	350	299
with Library Source Code	550	469
Various Support Utilities	CALL	CALL
Flash-up Windows *by Software Bottling*	90	79
GraphiC *Color version by Sci Endeavors*	350	282
GraphiC *Mono version by Sci Endeavors*	280	209
GRAFLIB *by Sutrasoft*	175	159
HALO *Graphics by Media Cybernetics*	300	205
HALO Development Pkg *for Microsoft*	595	389
The HAMMER *by OES Systems*	195	129
MetaWINDOWS *No Royalties*	185	109
MetaFONTS	80	58
MetaWINDOWS/Plus *by Metagraphics*	235	185
MetaFONTS/Plus	235	185
PANEL *by Roundhill Computer Systems*	295	215
PC Lint *by Gimpel Software*	139	99
PLOTHI *by Sutrasoft*	175	159
PLOTHP *by Sutrasoft*	175	159
Professional C Windows *by Washburn* New	CALL	CALL
Scientific Subroutine Library *by Peerless*	175	128
screenplay *for C by Flexus*	175	129
Vitamin C *by Creative Programming*	225	158
VC Screen *Forms Designer*	100	79
Zview *by Data Management Consultants*	245	169

cobol language

EASY SCREEN *by Retail Mgmt Systems* New	225	199
Micro Focus COBOL Workbench	4000	CALL
Micro Focus Level II COBOL	1500	CALL
COGRAPHICS	250	CALL
COMATH	200	CALL
FORMS-2	300	CALL
Level II Animator	900	CALL
Level II SOURCEWRITER	2000	CALL
Micro Focus Level II COBOL *for Novell*	2000	CALL
Micro Focus Professional COBOL	3000	CALL
Multi-user Runtime *for PC Network*	500	CALL
Microsoft COBOL *See Microsoft Section*		
Realia COBOL	995	783
REALCICS New	995	783
REALMENU New	150	119
RM/COBOL *by Ryan-McFarland*	950	CALL
RM/COBOL 85 *by Ryan-McFarland*	1250	CALL
screenplay *for COBOL by Flexus*	175	129

debuggers & profilers

386 DEBUG *Cross Debugger by Phar Lap*	195	129
Advanced Trace-86 *by Morgan Computing*	175	115
CI Probe *by Computer Innovations*	225	CALL
Codesifter *Profiler by David Smith*	119	85
Codesmith-86 *by Visual Age*	145	98
DSD86 *by Soft Advances*	70	61
DSD87 *by Soft Advances*	100	79
MiniProbe *by Atron*	395	CALL

Periscope I *with Board by Periscope*	345	289
Periscope II *with NMI Breakout Switch*	175	139
Periscope II-X *Software only*	145	105
The PROFILER *with Source Code by DWB*	125	89
The WATCHER *Profiler by Stony Brook*	60	51

dos utilities

Command Plus *by ESP Software* New	80	69
FANSI-CONSOLE *by Hersey Micro*	75	62
MKS Toolkit *with vi Editor by MKS*	139	99
Norton Commander *by Peter Norton*	75	55
Scroll & Recall *by Opt-Tech Data*	69	59
Taskview *by Sunny Hill Software*	80	56

entelekon products

C Dynamo New	250	209
C Dyanamo Screen Painter New	130	109
C Function Library	130	109
Dyanamo Data Entry New	130	109
Power Windows	130	109
Superfonts for C	50	43
Screen Painter & Data Entry New	180	150

essential products

C Essentials *by Essential Software*	100	75
C Utility Library	185	119
Essential Comm Library *with Debugger*	250	189
Essential Comm Library *Software Only*	185	125
Breakout Debugger *Any language*	125	89
Essential Graphics	250	183

forth language

CFORTH *Native Code Compiler by LMI*	300	229
Forth/83 Metacompiler *Specify Target*	750	599
PC/Forth *by Laboratory Microsystems*	150	109
PC/Forth+ *by Laboratory Microsystems*	250	199
Advanced Color Graphics Support	100	74
Enhanced Graphics Support	200	148
Intel 8087 Support	100	74
Interactive Symbolic Debugger	100	74
Native Code Optimizer	200	148
Software Floating Point	100	74
UR/Forth *by LMI*	350	279
Object Module Libraries	500	395
Source Code License	1500	995

fortran language

50 MORE: FORTRAN *by Peerless Engr*	125	95
ACS Time Series *Alpha Computer Service*	495	389
Essential Graphics *by Essential Software*	250	183
For-Winds *Alpha Computer Service*	90	69
Forlib-Plus *Alpha Computer Service*	70	44
FORTLIB *by Sutrasoft*	95	85
FORTRAN Addendum *by Impulse Engr*	95	85
FORTRAN Addenda *by Impulse Engr*	165	138
GRAFLIB *by Sutrasoft*	175	159
HALO *by Media Cybernetics*	300	205
I/O PRO *by MEF Environmental*	149	129
Microcompatibles Combo Package	240	215
Grafmatic	135	117
Plotmatic	135	117
Microsoft FORTRAN *w/CodeView* New Version	450	269
No Limit *by MEF Environmental*	129	115
Numerical Analyst *by MAGUS* New	295	CALL
PANEL *Screen Designer by Roundhill*	295	215
PLOTHI *by Sutrasoft*	175	159
PLOTHP *by Sutrasoft*	175	159
RM/FORTRAN *Ryan-McFarland*	595	CALL
RTC PLUS *Fortran to C by Cobalt Blue* New	325	CALL
Scientific Subroutine Lib *by Peerless*	175	128
Statistician *Alpha Computer Service*	295	245
Statlib.GL: *by PSI/Systems* New	295	239
Statlib.TSF: *by PSI/Systems* New	295	239
Strings & Things *Alpha Computer Service*	70	51

greenleaf products

Greenleaf Comm Library	185	125
Greenleaf Data Windows	225	157
with Source Code	450	289
Greenleaf Functions	185	125

help utilities

HELP/Control *by MDS*	125	99
On-line Help *from Opt-Tech* New Version	149	99
SoftScreen/HELP *by Dialectic Systems*	195	149

lattice products

Lattice C Compiler *from Lattice*	500	269
with Library Source Code	900	495
C Cross Reference Generator	50	37
with Source Code	200	139
C-Food Smorgasbord *Function Library*	150	95
with Source Code	300	179
C-Sprite *Source Level Debugger*	175	129
Curses *Screen Manager*	125	89
with Source Code	250	174
dBC II *Specify dBC II or dBC III*	250	169
with Source Code	500	356
dBC III Plus New	750	CALL
with Source Code New	1500	CALL
LMK *Make Facility*	195	138
RPG II Combo *All three items below*	1100	875
RPG II Compiler *No Royalties*	750	625
RPG II SEU *Screen Entry Utility*	250	199
RPG II Sort/Merge	250	199
RPG II Screen Design Aid Utility	350	309
SecretDisk *File Encryption Utility*	120	89
SideTalk *Resident Communications*	120	89
SSP/PC *Scientific Subroutine Library*	350	269
Text Management Utilities	120	89
TopView Toolbasket *Function Library*	250	178
with Source Code	500	356

microport products

System V/AT *by Microport Systems*	499	429
Runtime System *(Operating System)*	199	189
Software Development System	199	189
Text Preparation System	199	189
User Upgrade *3 to Unlimited Users*	CALL	CALL

Circle **no. 129** on reader service card.

microsoft products

Microsoft BASIC Interpreter *for XENIX*	350	209
Microsoft C *with CodeView*	450	269
Microsoft COBOL Compiler	700	429
for XENIX	995	609
Microsoft COBOL Tools *with Debugger*	350	194
for XENIX	450	289
Microsoft FORTRAN *w/CodeView* ... *New Version*	450	269
for XENIX	695	419
Microsoft Learning DOS	50	36
Microsoft LISP *Common LISP*	250	149
Microsoft MACH 10 *w/Mouse & Windows*	549	369
Microsoft MACH 10 *Board only*	399	279
Microsoft Macro Assembler	150	93
Microsoft Mouse *Bus Version*	175	114
Microsoft Mouse *Serial Version*	195	124
Microsoft muMath *Includes muSIMP*	300	179
Microsoft Pascal Compiler	300	179
for XENIX	695	419
Microsoft QuickBASIC Compiler	99	63
Microsoft Sort	195	125
Microsoft Windows	99	63
Microsoft Windows Development Kit	500	299

modula-2 products

IOTools *by Rhodes Associates* ... *New*	80	69
with Source Code ... *New*	950	CALL
MODULA-2 Apprentice Pkg by LOGITECH ... *New*	99	79
MODULA-2 Magic Pkg by LOGITECH ... *New*	99	79
MODULA-2 ROM Pkg & Cross RT Debugger *New*	299	239
MODULA-2 Window Pkg by LOGITECH ... *New*	49	39
MODULA-2 Wizard's Pkg by LOGITECH ... *New*	199	159
REPERTOIRE *for MODULA-2 by PMI*	89	79
Object Code Only	19	15

mouse products

LOGIMOUSE BUS *with PLUS Pkg by LOGITECH*	139	115
with PLUS & PC Paintbrush	189	149
with PLUS & CAD Software	209	175
with PLUS & CAD & Paint	239	195
LOGIMOUSE C7 *Specify Connector*	99	83
with PLUS Package	119	98
with PLUS & PC Paintbrush	169	134
with PLUS & CAD Software	189	153
with PLUS & CAD & Paint	219	179

other languages

CCS MUMPS *Single-User by MGlobal*	60	51
CCS MUMPS *Single-User/Multi-Tasking*	150	129
CCS MUMPS *Multi-User*	450	359
Janus/ADA C Pak *by R&R Software*	95	84
Janus/ADA D Pak *by R&R Software*	900	769
Janus/ADA ED Pak *by R&R Software* ... *New*	395	CALL
Marshal Pascal *by Marshal Language Systems* *New*	189	155
Personal REXX *by Mansfield Software*	125	99
SNOBOL4+ *by Catspaw*	95	80

other products

Dan Bricklin's Demo Pgm *Software Garden*	75	57
Disk Optimizer *by Softlogic Systems* ... *New*	60	55
FASTBACK *by 5th Generation Systems*	179	133
Instant Replay *by Nostradamus*	90	79
Net-Tools *by BC Associates* ... *New*	149	129
OPT-Tech Sort *by Opt-Tech Data Proc*	149	99
VTEK *Term Emulator by Sci Endeavors*	150	129

phoenix products

Pasm86 *Macro Assembler Version 2.0*	195	108
Pdisk *Hard Disk & Backup Utility*	145	89
Pfantasy Pac *Phoenix Combo* ... *Sale*	1295	799
Pfinish *Execution Profiler* ... *Sale*	395	209
Pfix86plus *Symbolic Debugger* ... *Sale*	395	209
PforCe *Comprehensive C Library* ... *Sale*	395	209
PforCe++ *Library for Guidelines C++* ... *Sale*	395	209
Plink86plus *Overlay Linker* ... *Sale*	495	279
Pmaker *Make Utility*	125	78
Pmate *Macro Text Editor*	195	108
Pre-C *Lint Utility*	295	154
Ptel *Binary File Transfer Program*	195	108

polytron products

PolyBoost *The Software Accelerator*	80	64
PolyLibrarian *Library Manager*	99	73
PolyLibrarian II *Library Manager*	149	109
PolyMake *UNIX-like Make Facility*	149	109
PolyShell	149	109
Polytron C Beautifier	50	42
Polytron C Library I	99	72
Polytron PowerCom *Communications*	139	105
PolyWindows Products *All Varieties*	CALL	CALL
PolyXREF *Complete Cross Ref Utility*	219	169
PolyXREF *One language only*	129	99
PVCS Corporate *Version Control System* ... *New*	395	309
PVCS Personal ... *New*	149	109

program mgmt utilities

Compact Source Print *by Aldebaran*	55	44
Interactive EASYFLOW *by Haventree*	150	125
PrintQ *by Software Directions*	89	84
Quilt Computing Combo Package	199	159
QMake *Program Rebuild Utility*	99	79
SRMS *Software Revision Mgmt Sys*	125	109

Source Print *by Aldebaran Labs*	75	59
TLIB *by Burton Systems Software*	100	89
Tree Diagrammer *by Aldebaran Labs*	55	49

raima products

dbQUERY *Single-User Query Utility*	195	129
Single-User with Source Code	495	389
Multi-User	495	389
Multi-User with Source Code	990	799
dbVISTA *Single-User DBMS*	195	129
Single-User with Source Code	495	389
Multi-User	495	389
Multi-User with Source Code	990	799

sco products

Complete XENIX System V *by SCO*	1295	999
Development System	595	499
Operating System *Specify XT or AT*	595	499
Text Processing Package	195	144
Networks for XENIX *by SCO*	595	495
SCO Professional *Lotus clone for XENIX*	795	595

softcraft products

Btrieve *ISAM Mgr with No Royalties*	245	184
Xtrieve *Query Utility*	245	184
Report Option *for Xtrieve*	145	99
Btrieve/N *for Networks*	595	454
Xtrieve/N	595	454
Report Option/N *for Xtrieve/N*	345	269

solution systems products

APT *Active Prolog Tutor*	65	CALL
Brief & dBrief Combo	250	CALL
Brief *Programmer's Text Editor*	195	CALL
dBrief *Customizes Brief for dBase III*	95	CALL
C Screen Editor	75	CALL
C ToolSet	95	CALL
Faster C	95	CALL
Prolog-86	125	CALL
Prolog-86 Plus	250	CALL
Security Library	125	CALL
with Source Code	250	CALL
TransLISP	95	CALL
TransLISP PLUS ... *New*	195	CALL
ZAP *Communications*	95	CALL

text editors

Brief *from Solution Systems*	195	CALL
Epsilon *Emacs-like editor by Lugaru*	195	147
KEDIT *by Mansfield Software*	125	98
Micro/SPF *by Phaser Systems* ... *New*	175	139
PC/VI *by Custom Software Systems*	149	99
SPF/PC *by Command Technology Corp* *New Version*	245	175
Vedit *by CompuView*	150	98
Vedit Plus *by CompuView*	185	128

turbo pascal utilities

ALICE *Interpreter by Software Channels*	95	66
DOS/BIOS & Mouse Tools *by Quinn-Curtis* ... *New*	75	67
Flash-up Windows *by Software Bottling*	90	79
MetraByte D/A Tools *by Quinn-Curtis* ... *New*	100	89
Science & Engrg Tools *by Quinn-Curtis* ... *New*	75	67
Screen Sculptor *by Software Bottling*	125	91
screenplay *for Turbo Pascal by Flexus*	100	79
Speed Screen *by Software Bottling* ... *New*	125	91
System Builder *by Royal American*	100	CALL
IMPEX *Query Utility* ... *New*	75	CALL
Report Builder	75	CALL
TDebugPLUS *by TurboPower Software*	60	49
Turbo EXTENDER *by TurboPower Software*	85	64
Turbo Professional *by Sunny Hill*	70	45
TurboHALO *from IMSI*	129	98
TurboPower Utilities *by TurboPower*	95	78
TurboRef *by Gracon Services*	50	45
TURBOsmith *Debugger by Visual Age*	69	45
TurboWINDOW *Graphics/Windows by MetaGraphics*	80	58

wendin products

Operating System Toolbox	99	75
PCNX *Operating system*	99	75
PCVMS *Similar to VAX/VMS*	99	75
XTC *Text Editor with Pascal source*	99	75

xenix/unix products

Btrieve *ISAM File Mgr by SoftCraft*	595	454
C-terp *by Gimpel, Specify compiler*	498	379
c-tree *ISAM Mgr by FairCom*	395	315
dbVISTA *See Raima Section*		
dBx *with Library Source by Desktop AI*	550	489
DOSIX *Console Version by Data Basics*	399	CALL
DOSIX *User Version by Data Basics*	199	CALL
Micro Focus Level II Compact COBOL	1000	CALL
Forms-2	400	CALL
Level II ANIMATOR	600	CALL
Microport Products *See Microport Section*		
Microsoft Products *See Microsoft Section*		
PANEL *Screen Designer by Roundhill*	625	535
REAL-TOOLS *Binary Version by PCT*	149	89
Library Source Version	399	289
Complete Source Version	499	369
RM/COBOL *by Ryan-McFarland*	1250	CALL
RM/COBOL '85 *by Ryan-McFarland*	CALL	CALL
RM/FORTRAN *by Ryan-McFarland*	750	CALL
SCO Products *See SCO Section*		

LOWEST PRICES

Since this ad is prepared in advance of publication, some of our current prices may be lower than what's advertised here. Call for latest pricing.

FREE SHIPPING

Orders within the USA (including Alaska & Hawaii) are shipped FREE via UPS. Express shipping is available at the shipping carrier's standard rate with no rush fees or handling charges. To avoid delays when ordering by mail, please call first to determine the exact cost of express shipping.

CREDIT CARDS

VISA and MasterCard are accepted at no extra cost. Your card is charged when your order is shipped. Mail orders please include credit card expiration date and telephone number.

CODs AND POs

CODs and Purchase Orders are accepted at no extra cost. POs with net 30-day terms are available to qualified US accounts only.

FOREIGN ORDERS

Shipping charges for foreign and Canadian orders are based on the shipping carrier's standard rate. Since rates vary between carriers, please call or write for the exact cost. Foreign orders (except Canada), please include an additional $10 for customs form preparation. All payments must be made with US funds drawn on a US bank. Please include your telephone number when ordering by mail. Due to government regulations, we cannot ship to all countries.

VOLUME ORDERS

Call for special pricing.

SOUND ADVICE

Our knowledgeable technical staff can assist in comparing products, answer technical questions and send you detailed product information tailored to your needs.

30-DAY GUARANTEE

Most of our products come with a 30-day documentation evaluation period or 30-day return guarantee. Please note that some manufacturers restrict us from offering guarantees on their products. Call for more information.

CALL TOLL-FREE

US	800-336-1166
CANADA	800-225-1166
OHIO & ALASKA (Call Collect)	216-877-3781
FOREIGN	216-877-3781
CUSTOMER SERVICE	216-877-1110

Hours: Weekdays 8:30 AM to 8:00 PM EST.

Ohio customers please add 6% state sales tax.

Prices are subject to change without notice.

Call or write for our FREE comprehensive price guide.

136 SUNNYSIDE STREET HARTVILLE, OHIO 44632

programmer's connection

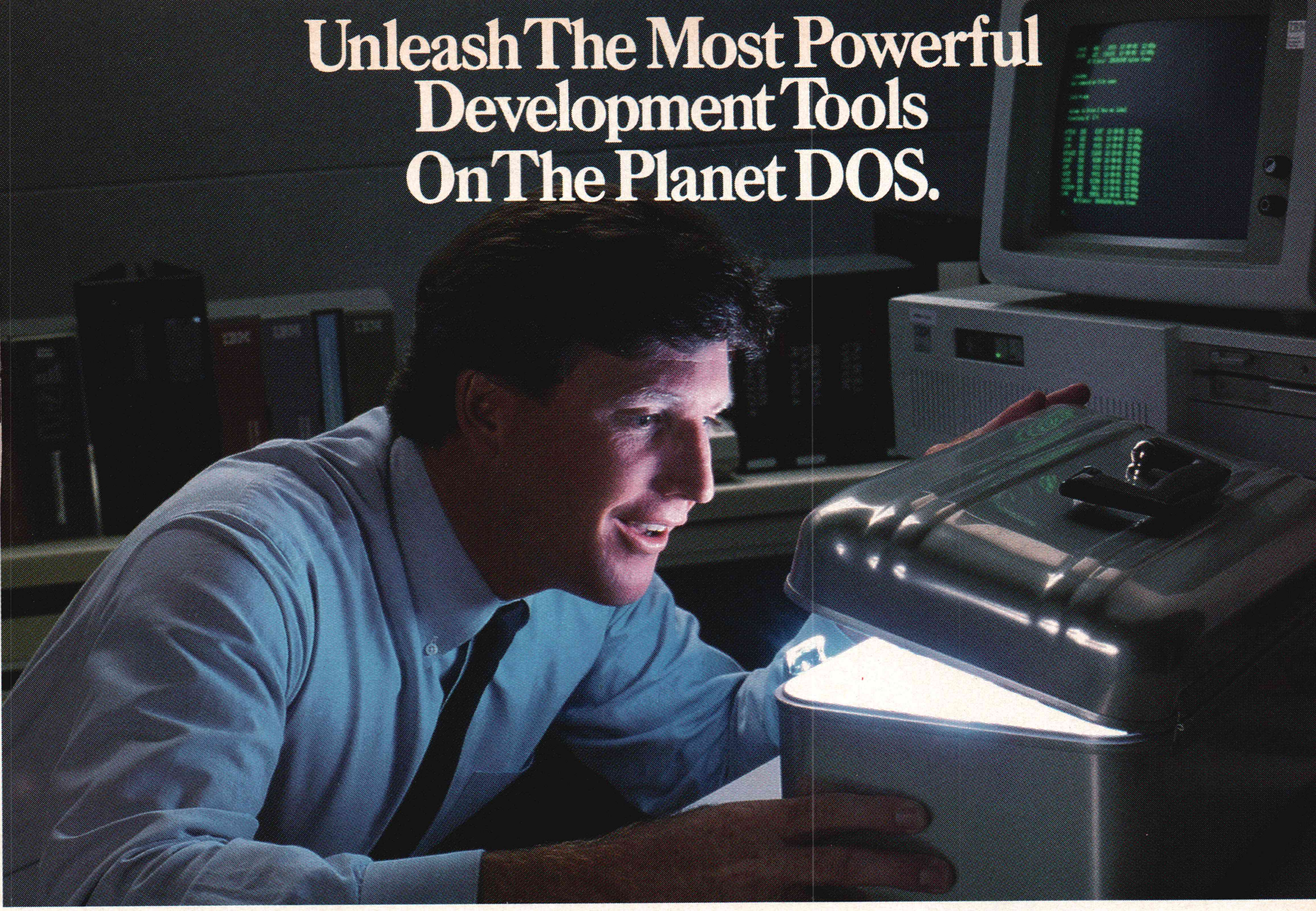
Circle **no. 129** on reader service card.

Listing One

(Listing continued, text begins on page 42.)

```
RETURN

Prove.long.legs:
  current.fact=long.legs:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.dark.spots:
  current.fact=dark.spots:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.tawny.color:
  current.fact=tawny.color:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.bird:
  current.fact=bird:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.feathers:GOSUB Prove.flies.and.lays.eggs
    number.of.or.clause.components=2
      OR.COMPONENT(1)=feathers
        AT.FACTOR.FOR.OR.COMPONENT(1)=1
      OR.COMPONENT(2)=flies.and.lays.eggs
        AT.FACTOR.FOR.OR.COMPONENT(2)=.8
    GOSUB Compute.or.clause.cf
  GOSUB Deduce
RETURN

Prove.ungulate:
  current.fact=ungulate:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.mammal.and.hoofs
  GOSUB Prove.mammal.and.chews.cud
    number.of.or.clause.components=2
      OR.COMPONENT(1)=mammal.and.hoofs:
        AT.FACTOR.FOR.OR.COMPONENT(1)=.85
      OR.COMPONENT(2)=mammal.and.chews.cud
        AT.FACTOR.FOR.OR.COMPONENT(2)=.8
    GOSUB Compute.or.clause.cf
  GOSUB Deduce
RETURN

Prove.carnivore:
  current.fact=carnivore:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.eats.meat:GOSUB Prove.teeth.claws.eyes
    number.of.or.clause.components=2
      OR.COMPONENT(1)=eats.meat
        AT.FACTOR.FOR.OR.COMPONENT(1)=.85
      OR.COMPONENT(2)=teeth.claws.eyes
        AT.FACTOR.FOR.OR.COMPONENT(2)=1
    GOSUB Compute.or.clause.cf
  GOSUB Deduce
RETURN

Prove.mammal:
  current.fact=mammal:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.has.hair:GOSUB Prove.gives.milk
    number.of.or.clause.components=2
      OR.COMPONENT(1)=has.hair:AT.FACTOR.FOR.OR.COMPONENT(1)=.85
      OR.COMPONENT(2)=gives.milk:AT.FACTOR.FOR.OR.COMPONENT(2)=.8
    GOSUB Compute.or.clause.cf
  GOSUB Deduce
RETURN

Prove.has.hair:
  current.fact=has.hair:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.gives.milk:
  current.fact=gives.milk:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.eats.meat:
  current.fact=eats.meat:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.teeth.claws.eyes:
  current.fact=teeth.claws.eyes:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.pointed.teeth:GOSUB Prove.claws
  GOSUB Prove.front.eyes
    number.of.and.clause.components=3
      AND.COMPONENT(1)=pointed.teeth:AND.COMPONENT(2)=claws
      AND.COMPONENT(3)=front.eyes
    at.factor.for.and.clause=.85
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN

Prove.mammal.and.hoofs:
  current.fact=mammal.and.hoofs:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.mammal:GOSUB Prove.hoofs
    number.of.and.clause.components=2
      AND.COMPONENT(1)=mammal:AND.COMPONENT(2)=hoofs
    at.factor.for.and.clause=.8
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN

Prove.mammal.and.chews.cud:
  current.fact=mammal.and.chews.cud:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.mammal:GOSUB Prove.chews.cud
    number.of.and.clause.components=2
      AND.COMPONENT(1)=mammal:AND.COMPONENT(2)=chews.cud
    at.factor.for.and.clause=.8
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN

Prove.feathers:
  current.fact=feathers:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.flies.and.lays.eggs:
  current.fact=flies.and.lays.eggs:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Prove.flies:GOSUB Prove.lays.eggs
    number.of.and.clause.components=2
      AND.COMPONENT(1)=flies:AND.COMPONENT(2)=lays.eggs
    at.factor.for.and.clause=1
    GOSUB Compute.and.clause.cf
  GOSUB Deduce
RETURN

Prove.lays.eggs:
  current.fact=lays.eggs:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.flies:
  current.fact=flies:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.chews.cud:
  current.fact=chews.cud:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.hoofs:
  current.fact=hoofs:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.front.eyes:
  current.fact=front.eyes:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.claws:
  current.fact=claws:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN

Prove.pointed.teeth:
  current.fact=pointed.teeth:GOSUB Test.fact.for.human.input
  IF leave=yes THEN RETURN
  GOSUB Deduce
RETURN
```

End Listing



UNIFY DBMS/DOS. The UNIX World Leader Brings A New Dimension To DOS Application Development.

What happens as the DOS world expands? As a new generation of hardware takes over? As networking becomes more important? The potential is enormous. But until now, the tools to achieve it have been limited.

Now a leader from another world unleashes that potential: UNIFY® DBMS. The leading relational DBMS in the UNIX™ world. And now, the most advanced set of application development tools in the DOS world.

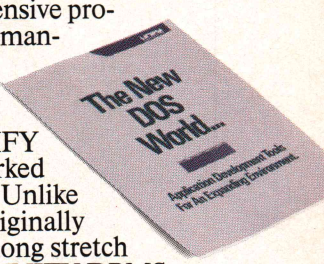
With UNIFY DBMS, DOS developers have new power to build more sophisticated applications than ever before possible.

The power to write high performance "C" programs that will access the data base, using Unify's Direct Host Language Interface.

The power of an industry standard query language—SQL.

The power of unmatched speed in production applications. Only UNIFY DBMS is specifically engineered for transaction throughput. With unique performance features like PathFinder™ Architecture multiple access methods, for the fastest possible data base access.

The power of comprehensive program development and screen management tools. Plus a state-of-the-art fourth generation report-writer.



What's more, with UNIFY DBMS, the potential of networked applications becomes a reality. Unlike DBMS systems which were originally single-user (and which have a long stretch to accommodate more users), UNIFY DBMS is a *proven* multi-user system.

And because UNIFY DBMS/DOS is the best of two worlds, it offers you the most powerful benefit of all: DBMS applications that can grow as your needs grow. From single user DOS. To networked DOS. To multi-user UNIX. All without changing your applications.

Call the Unify Information Hotline for our free booklet: The New DOS World. (503) 635-7777

UNIFY
CORPORATION
4000 Kruse Way Place
Lake Oswego, OR 97034

Circle **no. 332** on reader service card.

Listing Nineteen *(Text begins on page 130.)*

```
#include <stdio.h>
#include <hash.h>
#include "nr.h"

/*------------------------------------------------------------
 *
 *      NRMAC.C: macro, diversion, & trap support for nr
 *
 *      Copyright (c) 1987,  Allen I. Holub.
 *
 *      This module holds routines for mainpulating and
 *      accessing macros and strings. In addition the line
 *      trap mechanism is implemented here.
 *
 *------------------------------------------------------------
 * Macros are kept in a hash table. If they are smaller
 * than MAXMBUF characters, they are stored in memory; else
 * they're stored on the disk. When a macro gets too big it
 * is put into a file called XXYY.mac where XX is the first
 * character in the name represented as two hex digits, YY
 * is the second character.
 *
 * VERT is the height of the current diversion, accessable as
 * \n(.v It is used three places other than this module:
 *
 * 1) It's incremented after every line is output in nrout.c
 * 2) It's used to spring diversion traps in nrout.c
 * 3) It's used when setting a diversion trap in nrprocs.c
 *
 * it's value is equal to \n(nl if no diversion is active.
 * HEIGHT is the height of the most recently completed
 * diversion accessable as \n(dn. It's set equal to VERT
 * when a diversion is closed. It's not used anywhere by nr.
 *------------------------------------------------------------
 *      MACRO-RELATED DEFINES:
 */

typedef struct _macro_
{
    char mode;  /* Open mode: 'r'  'w'  'a'  0=not open     */
    FILE *fd;   /* Fd of macro on disk or 0 if in memory    */
    char *buf;  /* Pointer to buffer or 0 if macro on disk */
    char *ptr;  /* If buf valid, pointer to the             */
                /* last valid character. (current char if   */
                /* writing).                                */
    int vert;
    int width;  /* Macro size in lines and macro width in   */
                /* characters. These fields are only        */
                /* defined if the macro is a diversion.     */
}
MACRO;

typedef UCHAR   LTRAP[4];

static  LTRAP       Linetrap[ MAXLTRAP+1 ];
static  HASH_TAB    *Macros = NULL;

/*------------------------------------------------------------
 *      DIVERSION-RELATED DEFINES:
 */

typedef struct
{
    FILE *ofile;         /* Output file of previous level  */
    int  isdiv;          /* 1 if file is a diversion       */
    int  divtrap;        /* Diversion trap                 */
    char dtrap_name[2];  /* Invoke when divtrap reached    */

    int  width;          /* width of current diversion.    */
    int  vert;           /* Vertical place in current div  */
}
DIV;

#define MAXDIV  8        /* Maximum diversion nesting level */

DIV     Dstack[MAXDIV];  /* Diversion environment stack     */
int     Dsp = MAXDIV;    /* Diversion stack pointer (index) */

/*------------------------------------------------------------
 *  Function prototypes for external routines not declared
 *  in a .h file
 */

extern void err        ( char*, ...              ); /* nrout.c */
extern int  getline    ( char*,int,int(*)()      ); /* nrinp.c */
extern void process    (FILE*,char*,int,char**); /* nrinp.c */
extern char *skipspace( char*, int               );
extern char *skipto    ( int, char*, int         );
extern char *getenv    ( char*                   );

/*-----------------------------------------------------------*/
/*  Function prototypes for routines in this module         */

/* MACRO-RELATED (primitives):  */

/*global*/  int    mgetc         (MACRO *          );
/*global*/  void   mwrite        (MACRO *, char   *);
/*global*/  void   mputc         (int,      MACRO *);
/*global*/  void   munlink       (char *           );
/*global*/  MACRO *mopen         (char*, char*     );
/*global*/  void   mclose        (MACRO*           );

/* MACRO-RELATED (high level):  */

/*global*/  char *expandstr      (char *,char *,int);
/*global*/  int   expand_macro   (char *);
/*global*/  int   mcreate        (char *,char *);
/*global*/  int   mappend        (char *,char *);
/*global*/  void  mac_clean      (void);
/*global*/  int   screate        (char *,char *);
/*global*/  int   sappend        (char *,char *);
/*global*/  int   printm         (void);

/* DIVERSION-RELATED */

/*global*/  int   dcreate        (char *);
/*global*/  int   dappend        (char *);
/*global*/  int   endiv          (void);

/* TRAP-RELATED */

/*global*/  int   set_linetrap   (char *,int );
/*global*/  int   movetrap       (char *,int ,int );
/*global*/  int   pr_traps       (void);
/*global*/  int   do_divtrap     (void);
/*global*/  int   do_linetrap    (int );
/*global*/  int   distance       (void);

/* USED LOCALLY */

/*local */  void  delm           (char*, MACRO*  );
/*local */  char  *fname         (char*          );
/*local */  void  swrite         (MACRO*,char*   );
/*local */  void  prnt           (char*, MACRO*  );
/*local */  int   pushdiv        (MACRO*         );
/*local */  MACRO *popdiv        (void           );
/*local */  int   findtrap       (char*          );

/*----------------------------------------------------------*/

static  char    *fname( name )
char    *name;
{
    /* Create a unique file name for a macro temporary file.
     * The name is volitile (it won't be preserved between
     * fname() calls. If a TMP environment exists, its
     * prefixed to the name.
     */

    static char buf[ 80 ];
    char                *env;

    if( !(env = getenv("TMP")) )
            env = "";

    sprintf(buf, "%s%02x%02x.mac", env, name[0] & 0xff,
                                        name[1] & 0xff);
    return buf;
}

/*----------------------------------------------------------*/

MACRO   *mopen( m_name, how )
char    *m_name;
char    *how;
{
    /* Open the macro "m_name" in the specified mode. Mode
     * may be "r" "w" or "a". If the macro doesn't exist it
     * is created. An open for write will delete the contents
     * of the macro if they exist. Only the first two
     * characters of the name mean anything. Return 0 on
     * error or a pointer to the macro on success.
     */

    register int        existing ;
    register MACRO      *pnode ;
    char                *name ;

    if( *m_name == '\0')
            return( (MACRO *)0 );

    if( !Macros )                          /* Create macro table   */
            Macros = maketab( 127 );       /* if it doesn't exist. */

    name = fname( m_name );               /* Convert macro name to */
                                          /* associated file name. */

    pnode    = (MACRO *) findsym(Macros, m_name);
    existing = pnode != NULL;

    if( !pnode )
    {
        pnode = (MACRO*) addsym(Macros, m_name, sizeof(MACRO));
    }
    else if( pnode->mode )
    {
        err("May not access .%2s macro recursively\n", m_name);
        return( (MACRO *)0 );
    }

    switch(  pnode->mode = *how  )
    {
    case 'a':
            if( existing  &&  pnode->fd )
                pnode->fd = fopen( name, "ab" );

            break;

    case 'w':
            if( existing )
            {
                /* If the macro already exists, truncate it's
                 * buffer, or buffer file, to zero
                 * length.
                 */

                if( pnode->fd )
                        pnode->fd = fopen( name, "w" );
```

(continued on page 86)

Why Are So Many People Switching to Smalltalk/V?

Why are scientists, engineers, and professionals switching to Smalltalk/V? Because it lets them do amazing things on their PCs, with a Mac-like interface and an easy-to-use object-oriented language. And with Smalltalk/V you get an unsurpassed array of problem-solving tools. You can even personalize the entire system to suit your needs.

Smalltalk/V is the programmable environment that gives you total control of your computer and makes it what it was meant to be, a truly personal tool for your mind.

"Smalltalk/V is the highest performance object-oriented programming system available for PCs."

Dr. Piero Scaruffi,
Chief Scientist, Olivetti
Artificial Intelligence Center

"Smalltalk/V is an excellent buy and makes a good alternative to other programming languages for the development of complex applications."

Bill Wong, Director, PC Labs
PC Magazine

$99.00

Smalltalk/V

The Programmable Environment

"This is the real thing, folks. A super Smalltalk like this turns your PC into a hot workstation. It's fantastic . . . Highly recommended."

John C. Dvorak,
Contributing Editor, PC Magazine

"My background is in physical chemistry, not in programming. I like Smalltalk/V because I can use objects in the computer to represent objects in the physical world."

Dr. Paul Soper, Senior Specialist
E. I. du Pont de Nemours & Co.

"Smalltalk/V is a productive programming environment that allows us to quickly develop sophisticated medical applications."

Dr. Mike McCoy,
Dean for Instructional Computing
UCLA School of Medicine

"Smalltalk/V, with its visual interface and class structure, is a perfect way to simulate the complex interactions of natural systems."

Lee A. Graham, Research Assistant
Institute of Ecology, University of Georgia

"I solve problems quickly using Smalltalk/V because its classes and objects help me organize my thinking. And besides, it's fun to use."

Dr. Barry Fishman, Sr. Project Engineer
Hughes Aircraft Company

Other Smalltalk/V Features

- Object-oriented Prolog integrated with the Smalltalk environment
- Supports exploratory programming and prototyping
- Class hierarchy with inheritance creates highly re-useable source code
- Smalltalk source code included, with browser windows for easy access and modification
- Object-swapping creates a virtual memory on hard or RAM disk
- Bit-mapped graphics with bit and form editors
- A sophisticated source-level debugger
- Automatic change log for easy recovery from errors
- Powerful directory/file browser system for organizing DOS files
- Access to other languages and DOS functions
- 500 page manual with comprehensive tutorial and reference sections
- Optional add-on modules
  - RS-232 communications interface with UNIX™ and TTY windows
  - EGA color graphics
  - "Goodies" diskette, including multiprocessing, music, zoom, object loader, and more

Yes! I want to turn my PC into a hot workstation! Send me . . . .

☐ Smalltalk/V-The Programmable Environment	$99
☐ Communications Option	$49
☐ EGA Color Option	$49
☐ "Goodies" diskette	$49
Shipping and Handling	$______
CA residents add applicable sales tax	$______
TOTAL	$______

Shipping and Handling

U.S., Canada, Mexico	$ 5.00
Elsewhere	$15.00

I enclose ☐ Check ☐ Money Order
☐ Credit card information ☐ MC ☐ VISA

Number: ______
Expiration: ______
Signature: ______
Name: ______
Street Address: ______
City/State/Zip: ______
Phone: ______

NOT COPY PROTECTED, 60-DAY MONEY-BACK GUARANTEE
ON-LINE USER-SUPPORT CONFERENCE ON BYTE'S BIX™

Smalltalk/V requires DOS and 512K RAM on IBM PCs (including AT) or "compatibles," and a CGA, EGA, Toshiba T3100, Hercules, or AT&T 6300 graphic controller. A Microsoft or compatible mouse is recommended.

digitalk inc.
5200 West Century Boulevard
Los Angeles, CA 90045 (213) 645-1082

Circle **no. 127** on reader service card.

Listing Nineteen

(Listing continued, text begins on page 130.)

```
                else if( pnode->buf )
                {
                        free( pnode->buf );
                        pnode->buf = 0  ;
                        pnode->ptr = "" ;
                }
            }
            break;

    case 'r':
            if( !existing )
            {
                delsym( Macros, (BUCKET *) pnode );
                return( (MACRO *)0 );
            }

            /*  Position the buffer (or file) pointer to
             *  beginning of buffer (or file).
             */

            if( pnode->buf )
                    pnode->ptr  = pnode->buf ;

            else if( pnode->fd )
            {
                if( !(pnode->fd = fopen( name, "r") ))
                {
                    err("Can't open macro file: <%s>\n", name);
                    return( (MACRO *)0 );
                }
            }

            break;

    default:
            err("Internal errror: bad mopen mode\n");
            return( (MACRO *)0 );
    }

    return( pnode );
}

/*--------------------------------------------------------*/

void    mclose( mptr )
MACRO   *mptr;
{
        if( mptr )
        {
            if( mptr->fd )
                fclose( mptr->fd );     /* Close the file  */

            mptr->mode  = 0;
        }
}

/*--------------------------------------------------------*/

int     mgetc( mptr )
MACRO   *mptr;
{
     /* Read from a macro opened with a previous mopen call.
      * mgetc is called by getline which is called by
      * process(). It's also called by expandstr() which may
      * be called while expanding a macro. Bunches of
      * recursion, use a big stack. Return EOF at end of macro.
      */

     register int         rval;

     if( mptr->buf )
          rval = *mptr->ptr ? (int)(*(mptr->ptr)++) : EOF ;

     else if (mptr->fd )
          rval = ( mptr->fd ) ? getc( mptr->fd ) : EOF ;

     else
          rval = EOF ;

     return rval;
}

/*--------------------------------------------------------*/

static  void    swrite( mptr, buf )
MACRO   *mptr;
char    *buf;
{
        /* Write into a string. buf is a pointer to the
         * string itself and mptr is a pointer to the macro.
         */

        while( *buf )
                mputc(*buf++, mptr) ;

        mputc( 0, mptr );
}

/*--------------------------------------------------------*/

void    mwrite( mptr, term )
MACRO   *mptr;
char    *term;
{
     /* Write into a macro. Input is taken from the
      * current input file and put into the macro until
      * a line starting with "term" is encountered. Note
      * that the input is not modified ( escape sequences
      * are not expanded, etc.). Terminate on end of file
      * or on encountering term at the beginning of a
      * line. Mwrite is also used by the .ig command to
      * ignore input text. To do this, set mptr to 0.
      */

     char        buf[ MAXSTR ], *bp ;
     int not_eof ;

     while( not_eof = getline(buf,1,Ismacro ? mgetc : fgetc))
     {
          if( ! *term )          /* Terminate on .. at bol  */
          {
              if( buf[0] == '.'  &&  buf[1] == '.' )
                      break;
          }
          else if( *term == '\n' )
          {
              if( !*buf )         /* Terminate on ah empty line */
                  break;
          }
          else if( buf[0]=='.' &&  buf[1]==term[0] && buf[2]==term[1] )
              break;


          if( !mptr )           /* For the .ig command. Ignore */
                  continue;     /* input.                      */


          for( bp = buf ; *bp ;  mputc( *bp++, mptr ) )
                  ;

          mputc( '\n', mptr );  /* Getline doesn't buffer LF */
     }

     if( not_eof )
     {
          if( mptr )
              mputc( 0, mptr );
     }
     else
     {
          err("EOF encountered while writing to %s macro\n",
                                               symname(mptr));
          exit(1);
     }
}

/*--------------------------------------------------------*/

void    mputc( c, mptr )
MACRO   *mptr;
{
     char        *name;

     if( mptr->fd )
     {
          /* Macro is on the disk:
           * Don't write a terminating null to a file. This
           * simplifies our life when we append to a macro
           * in a file.
           */

          if( c )
              putc(c, mptr->fd );
     }
     else if( mptr->buf )
     {
          /* Macro is in memory: Write the character into
           * the buffer. Don't increment when we write a null
           * to make appending easier.
           */

          if( mptr->ptr - mptr->buf < MAXMBUF )
          {
              if( c )
                  *(mptr->ptr)++ = c ;
              else
                  *mptr->ptr = 0 ;
          }
          else
          {
              /*  Macro has grown too large. Create a disk
               *  file and write it out to there. Free the
               *  memory previously used by the macro.
               */

              name = fname( symname(mptr) );

              if( !(mptr->fd = fopen(name , "w")) )
              {
                  err("Can't open temporary macro file <%s>\n",
                                                          name);
              }
              else
              {
                  fwrite( mptr->buf, MAXMBUF, 1, mptr->fd );
                  free( mptr->buf );
                  mptr->buf = mptr->ptr = 0 ;

                  if(c)
                      putc(c, mptr->fd);

              }
          }
     }
     else     /* New macro, allocate a buffer then write */
     {
          if( mptr->buf = (char *) malloc( MAXMBUF ) )
          {
              mptr->ptr = mptr->buf ;
```

(continued on page 89)

Think of us as the Book of the Mind Club.

Circle **no. 125** on reader service card.

Listing Nineteen *(Listing continued, text begins on page 130.)*

```
          if( c )
               *(mptr->ptr)++ = c ;
          else
               *mptr->ptr = 0;
     }
     else
          err("Insufficient memory for macro\n");
  }
}

/*-------------------------------------------------------*/

void     munlink(m_name)
char     *m_name;
{
         /* Remove macro "m_name" if it exists. Return 0 if
          * the register didn't exist;  return 1 if it was
          * removed successfully.
          */

         register MACRO  *node;

         if( !(node = (MACRO *) findsym( Macros, m_name )) )
              err("Macro <%2.2s> doesn't exist\n", m_name );
         else
         {
              if( node->mode )
              {
                   err("May not remove active macro, aborting\n");
                   exit( 1 );
              }

              if( node->buf )
                   free( node->buf );

              else if( node->fd )
                   unlink( fname( m_name ) );

              delsym( Macros, (BUCKET *) node );
         }
}

/*-------------------------------------------------------*/

char     *expandstr( name, target, maxstr )
char     *name, *target;
{
     /* Expand str into the target string. Return the updated
      * target pointer, which won't be modified if the string
      * doesn't exist. In this last case print an error message.
      * Expand at most maxstr characters. Note that there's
      * a indirect recursion if the expanded sting contains
      * an escape sequence. expandstr() is called from
      * escape().
      */

     register MACRO      *mptr;
     register int        c  ;
     char                *p ;

     if( mptr = (MACRO *) mopen(name, "r") )
     {
          c = mgetc(mptr) ;

          while( c != EOF   &&   maxstr > 0 )
          {
               if( c != Esc )
               {
                    *target++ = c    ;
                    c = mgetc(mptr) ;
                    --maxstr;
               }
               else
               {
                    p = target;
                    c = escape(p, &target, 0, mgetc, mptr, maxstr);
                    maxstr -= (target - p);
               }
          }

          mclose( mptr );
     }

     return target;
}

/*-------------------------------------------------------*/

expand_macro( str )
char     *str;
{
     /* Expand the macro. The first word in str is the name.
      * Note that expand is called by process() which calls
      * expand. There can be some nasty recursion
      * going on if macro expansion is nested too far. On the
      * other hand the code needed to expand nested macros
      * is much cleaner. MAXNEST will help this a little.
      *
      * Return 0 if the macro doesn't exist of it the nest
      * level is too high, 1 otherwise.
      *
      * This routine is called recursively in the case of
      * nested macro expansions. Be careful with static
      * variables.
      */

     register int        i, onargs;
     register MACRO      *mptr;
```


(continued on next page)

C CHEST

Listing Nineteen

(Listing continued, text begins on page 130.)

```
    char            *macv[MAXARGS], *name ;
    static          nestlev = 0 ;
    if( nestlev < MAXNEST )
        ++nestlev;
    else
    {
        err( "Macro nesting too deep, ignoring <%s>\n", str);
        return 0;
    }

    name = str;                                 /* extract name    */
    str  = skipto( ' ', str, Esc );             /* Skip past name */
    if( *str )
            *str++ = '\0';                      /* terminate name & */
    str = skipspace(str, Esc);                  /* skip whitespace  */

    if( !( mptr = (MACRO *) mopen(name, "r") ) )
            return 0;

    /* Create the vector array pointing into the argument
     * array. Null terminate each argument string. Quoted
     * arguments are recognized.
     */

    for( i = 0 ;   *str && i < MAXARGS  ; i++ )
    {
            if( *str == '"' )
            {
                    macv[i] = ++str ;
                    str = skipto( '"', str, Esc );
            }
            else
            {
                    macv[i] = str ;
                    str = skipto( ' ', str, Esc );
            }

            if( *str )
            {
                    *str++ = 0;
                    str = skipspace(str, Esc);
            }
    }

    onargs = NARGS;             /* Set # args at current lev */
    NARGS  = i;

    while( i < MAXARGS )        /* Put null in unused args    */
            macv[i++] = "" ;

    process( (FILE *) mptr, symname(mptr), 1, macv );

    NARGS = onargs;             /* clean up                   */
    --nestlev;
    mclose( mptr );
    return 1;
}

/*------------------------------------------------------------*/

dump_mac( macro, file )
char    *macro, *file;
{
        /* Dump the indicated macro out to the indicated file.
         */

        MACRO           *mptr;
        FILE            *fp;
        register int    c;

        if( !(fp = fopen(file,"w")) )
                err("Can't open <%s> for output\n", file );

        else if( mptr = (MACRO *) mopen(macro, "r") )
        {
                while( (c = mgetc(mptr)) != EOF  )
                        putc(c, fp);

                mclose( mptr );
                fclose( fp   );
        }
}

/*------------------------------------------------------------*/

mcreate( name, term )
char    *name, *term;
{
        /* Create a macro. If it already exists, delete it
         * first.
         */

        register MACRO  *mptr;

        if( mptr = mopen( name, "w" ) )
        {
                mwrite( mptr, term );
                mclose( mptr );
        }
}

/*------------------------------------------------------------*/

mappend( name, term )
char    *name, *term;
{
        /* Append to an existing macro. Create it if it
         * doesn't exist.
         */
```

```
        register MACRO  *mptr;

        if( mptr = mopen( name, "a" ) )
        {
                mwrite( mptr, term );
                mclose( mptr );
        }
}

/*-----------------------------------------------------------*/

screate(name, str)
char    *name, *str;
{
        /*      Create a string. If it exist, delete it
         */

        register MACRO  *mptr;

        if( mptr = mopen( name, "w" ) )
        {
                swrite( mptr, str );
                mclose( mptr );
        }
}

/*-----------------------------------------------------------*/

sappend(name, str)
char    *name, *str;
{
        /* Append to an existing string. If it doesn't
         * exist, create it.
         */

        register MACRO  *mptr;

        if( mptr = mopen( name, "a" ) )
        {
                swrite( mptr, str );
                mclose( mptr );
        }
}

/*-----------------------------------------------------------*/

static  void    prnt( m_name, p )
char    *m_name;
MACRO   *p;
{
        FILE            *stream = NULL;
        int             len;
        char            str[80];

        if( p->buf )
                len = strlen( p->buf );

        else if( p->fd )
        {
            if( !(stream = fopen( fname(m_name), "rb" )))
            {
                err("Can't open %s\n", str );
                return;
            }

            len = filelength( fileno(stream) ) ;
        }

        printf("+-------- <%s> (%d chars in %s) --------+\n",
                m_name, len, stream ? "file" : "memory" );

#ifdef DEBUG
        printf("| mode=0x%x=<%c>, buf=0x%x, ptr=0x%x, fd=0x%x\n",
                p->mode, p->mode, p->buf, p->ptr, p->fd );

        printf("+-----------------------\n" );
#endif

        if( !stream )
                fputs( p->buf, stdout );
        else
        {
                while( fgets(str, 80, stream) )
                        fputs( str, stdout );

                fclose(stream);
        }

        putchar('\n');
}

/*-----------------------------------------------------------*/

printm()        /*      Print out all the macros        */
{
        register int lev;

        if( !Macros )
                err("**** There are no macros ****\n");
        else
        {
                ptab( Macros, prnt );
                printf( "\nThe end macro is <%s>\n",
                        *Endm ? Endm : "NONEXISTANT" );
        }
}

/*-----------------------------------------------------------*/

static void     delm( m_name, p )
char    *m_name;
MACRO   *p;
```

(continued on next page)

Listing Nineteen

(Listing continued, text begins on page 130.)

```
{
        /*      Delete disk file associated with macro */

        if( p->fd )
                unlink( fname( m_name ) );
}


void    mac_clean()
{
        /*      Delete all macros that are on the disk */

        if( Macros )
                ptab( Macros, delm );
}

/*----------------------------------------------------------------
 * Stuff to handle to diversions. It is too complicated to
 * use recursion here, mostly because you can be processing
 * a macro while you are diverting output (and have several
 * levels of nesting to boot. Modifying process() to handle
 * changes in both input and output prooved to be too
 * difficult. The easy thing to do is maintain a special
 * diversion stack on which we keep the various globals we
 * want to save when we change diversions. Pushdiv() and
 * popdiv() (below) do the stack maintenance.
 */

static  int     pushdiv( ndptr )
MACRO   *ndptr;
{
    register DIV        *div;

    if( Dsp <= 0 )
    {
            err("Diversion nesting too deep\n");
            return 0;
    }

    div = &Dstack[--Dsp];

#ifdef DEBUG
    printf("Opening diversion, saving at Dstack[%d]\n",
                                            Dsp);
#endif

    div->ofile              = Ofile  ;
    div->isdiv              = Isdiv  ;
    div->divtrap            = Divtrap;
    div->dtrap_name[0]      = Dtrap_name[0];
    div->dtrap_name[1]      = Dtrap_name[1];
    div->vert               = VERT;
    div->width              = Divwidth;

    Divwidth = 0;                   /* Width of current diversion */
    Ofile    = (FILE *)ndptr;       /* output macro pointer       */
    Isdiv       = 1 ;               /* Ofile points at a macro    */
    Divtrap     = -1 ;              /* No diversion trap set      */
    Dtrap_name[0] = 0 ;             /* Diversion trap has no name */

    return 1;
}

/*---------------------------------------------------------*/

static  MACRO   *popdiv()
{
    /* Restore the enviornment active before the most recent
     * push div call. Return a pointer to the diversion macro
     * or 0 if no enviornment to restore.
     */

    register MACRO      *rval;
    register DIV        *div;

    if( Dsp >= MAXDIV ) /* No diversion is active       */
            return 0;

#ifdef DEBUG
    printf("Closing diversion, poping Dstack[%d]\n", Dsp);
#endif

    div = &Dstack[Dsp++];

    /* Put back the old enviornment. Note that VERT is
     * initialized to 1 and incremented after every line
     * output to the diversion. This way \n(.d will be 1
     * on line 1 of the diversion. HEIGHT, however, is the
     * height of the most recent diversion (which will be
     * one less than VERT.
     */
    rval            = (MACRO *)Ofile;
    Ofile           = div->ofile    ;
    Isdiv           = div->isdiv    ;
    Divtrap         = div->divtrap  ;
    Dtrap_name[0]   = div->dtrap_name[0];
    Dtrap_name[1]   = div->dtrap_name[1];
    VERT            = div->vert;
    Divwidth        = div->width;

    return( rval );
}

/*---------------------------------------------------------*/

dcreate( name )
char    *name;
{
    /*      Create a diversion from scratch.
     */
```

```
    register MACRO  *mptr;

    if( mptr = mopen( name, "w" ) )
    {
        if( !pushdiv(mptr) )
            mclose( mptr );

        VERT     = 1; /* Vertical place in current div  */
        Divwidth = 0; /* Width of current diversion     */
    }
}

/*-----------------------------------------------------------*/

dappend( name )
char     *name;
{
        /* Open an existing diversion for appending. The
         * current diversion height and width number registers
         * will be changed to reflect this diversion.
         */

        register MACRO  *mptr;

        if( mptr = mopen( name, "a" ) )
        {
                if( !pushdiv(mptr) )
                        mclose( mptr );

                VERT     = mptr->vert +1 ;
                Divwidth = mptr->width   ;
        }
}

/*-----------------------------------------------------------*/

endiv()
{
    /* Close the most recently opened diversion
     * We must decrement VERT because it's incremented after
     * the final \n of the diversion is processed.
     * If no diversion is active, nothing is done.
     */

    register MACRO  *mptr;
    int             height, width ;

    height = VERT - 1 ;
    width  = Divwidth ;

    if( mptr = popdiv() )
    {
        mputc( 0, mptr );

        mptr->vert  = HEIGHT = height ;
        mptr->width = WIDTH  = width  ;

        mclose( mptr );
    }
}

/*-----------------------------------------------------------
 *
 *      Stuff to handle to traps:
 */

static  int     findtrap( name )
char    *name;
{
        /* Look for the trap associated with the macro
         * "name." Return an index if found, -1 if not.
         */

        register int    i;

        for( i =  MAXLTRAP+1 ; --i >= 0 ; )
                if( !strcmp(name, Linetrap[i]) )
                        return i;
        return -1;
}

/*-----------------------------------------------------------*/

set_linetrap( name, lnum )
char    *name;
int     lnum;
{
    /* Set a line trap that will execute the macro called
     * "name" when output line number "lnum" is passed. If
     * the name * is missing, clear the trap at the
     * indicated location.
     */

    register UCHAR      *lp;

    if( lnum < 0 )
        lnum += PGLEN ;

    lp = (UCHAR *)( Linetrap + lnum );

    if( lnum < 0   || lnum > MAXLTRAP )
    {
        err("line trap must be in the range 0 - %d\n",
                                                MAXLTRAP);
    }
    else if( !*name )
    {
        *lp = 0;
    }
    else
```

(continued on next page)

Listing Nineteen

(Listing continued, text begins on page 130.)

```
    {
            *lp++ = *name++ ;
            *lp++ = *name   ;
            *lp   = 0 ;
    }
}

/*------------------------------------------------------------*/

movetrap( name, where, isoffset )
char    *name;
{
    /* Deal with the .ch command. If "where" is 0
     * delete the trap else move it to the indicated
     * position. If "isoffset" then add "where" to
     * the current position. Note that it is not an
     * error to clear a non-existant trap.
     *
     * The first set_linetrap call deletes the
     * existing trap, the second one reinstalls it
     * at the new location.
     */

    register int        i;

    if( (i = findtrap(name)) >= 0 )
    {
        set_linetrap( "", i );

        if( where )
            set_linetrap( name, isoffset? i+where : where);
    }
}

/*------------------------------------------------------------*/

pr_traps()
{
    /*  Print all active traps: */

    register int        i, none = 1;

    printf("Line traps:\n");
    for( i = 0; i <= MAXLTRAP ; i++ )
    {
        if( Linetrap[i][0] )
        {
            if( none )
            {
                printf("execute:  on line:\n");
                none = 0;
            }

            printf("  %2.2s       %4d\n", Linetrap[i], i);
        }
    }

    if( none )
        printf( "There are no line traps set.\n");

    if( Divtrap != -1 )
        printf("Diversion trap <%s> set at line %d\n",
                                   Dtrap_name, Divtrap);
    if( Itrap != -1 )
        printf("Input line trap <%s> set at line %d\n",
                                   Itrap_name, Itrap);
}

/*------------------------------------------------------------*/

do_divtrap()
{
    /*  Spring the diversion trap  */

    if( !expand_macro( Dtrap_name ) )
            err("Can't spring .%2.2s from diversion trap\n",
                                          Dtrap_name );
}

/*------------------------------------------------------------*/

do_linetrap( lnum )
{
    /* Spring a line trap on line "lnum", if one exists */

    register UCHAR      *trap ;

    if( 0 <= lnum  &&  lnum <= MAXLTRAP )
    {
        trap = (UCHAR *)( Linetrap + lnum );

        if( *trap &&  !expand_macro(trap) )
            err("Can't spring trap for line %d (%2.2s).\n",
                                   (char *)lnum, trap );
    }
}

/*------------------------------------------------------------*/

distance()
{
    register char       *trap ;
    register int        line  ;

    /*   Compute distance from current line to next trap
     */

    line = OLINE ;                    /* -- Current output line */
```

(continued on page 98)

THE PROGRAMMER'S SHOP

Offers a 31 Day Money Back Guarantee on any product in this ad. Call Today.

COBOL PROGRAMMING PRODUCTIVITY!

screenplay™

SCREEN MANAGEMENT SYSTEM

SUBSTANTIALLY REDUCE APPLICATION DEVELOPMENT TIME!
Save valuable time by avoiding the tedious, time consuming process of writing screen handling source code. *screenplay's* easy-to-use panel painter allows you to create I/O panels, pop-up help windows or menu panels which you use in your program.

FLEXIBILITY IN PANEL PAINTING; FLEXIBILITY AT RUNTIME!
True screen handling flexibility is yours with *screenplay*, because you can override default panel settings to design practically any type of panel imaginable! *screenplay* continues to provide the flexibility you need during the operation of your program by allowing you to change just about any panel characteristic at runtime.

PROTOTYPE PANELS BEFORE YOU WRITE CODE!
With *screenplay*, you can prototype your draft screens before you write a single line of COBOL source code. These can then be reviewed with your boss or your customer for final approval, before you start writing source code.

MORE THAN ONE LINKING OPTION!
In addition, linking *screenplay's* runtime unit is your choice! You can link *screenplay* by interrupt or directly to your application. And if your compiler doesn't allow a direct link, you can take advantage of a dynamic load option for linking *screenplay* to your application.

FULL CONTROL OVER KEY ASSIGNMENT!
You can assign practically any keyboard key to serve as a specific cursor function and define exactly which keys will return control to your application. *screenplay* gives you the power to entirely reconfigure the keyboard for your program.

POWERFUL, ONE-STEP PANEL PAINTING
screenplay's panel painting process is a "one-step" approach. There's no need to go through a separate process to establish fields on your I/O panel. What's more, you can use any ASCII character in your *screenplay* panels. You also have full control over character attributes such as foreground and background color, intensity and blinking.

EASY PANEL FILE MANAGEMENT!
Panels are stored in an ASCII file which is compressed to save memory and disk space *screenplay's* Panel Management Facility allows you to easily copy panels across and within files, rename panels, delete panels, test and print panel details. You can even print an image of the panel for your documentation!

NO ROYALTIES / NOT COPY PROTECTED!
screenplay's panel control runtime unit, better known as The Panel Control Facility may be linked to your program without paying a dime in royalties. In addition, *screenplay* isn't copy protected to make it even easier-to-use. The Panel Control Facility allows you to control almost every panel characteristic by using parameters.

SUPPORTS MANY COBOL COMPILERS!
Supports IBM COBOL, Microsoft COBOL, Realia COBOL, Ryan-McFarland COBOL and Ryan-McFarland COBOL 8X; List Price $175.00, **OUR PRICE $155.00**

flexus — *PROFESSIONAL TOOLS FOR PROFESSIONAL PROGRAMMERS*

Circle **no. 301** on reader service card.

HELP

is at hand

HELP/Control™ — an on-line help system for the IBM-PC. **HELP/Control** includes **HELP/Runtime, HELP/Popup** and our help screen compiler.

With **HELP/Runtime**, a few simple subroutine calls add context sensitive on-line help to your application. **HELP/Runtime** includes tested interfaces for C (Microsoft and Lattice), Pascal (Microsoft and Turbo), IBM BASIC (Interpreter and Compiler), Microsoft FORTRAN, COBOL (IBM and Realia) and assembler.

Use our concise screen definition language to build your help files. You define the bold captions on your help screens and specify the links to other screens. If you have existing documentation files, it's easy to mark them up to get on-line quickly. You can put an entire user or reference manual on-line, completely accessible to the user at all times.

HELP/Popup provides memory resident access to your custom help screens.

The complete package (software, on-line manual, printed manual, and demo programs) costs $125.00 and includes a royalty-free license to add **HELP/Runtime** to your applications and a license to make 25 copies of **HELP/Popup.**

Circle **no. 303** on reader service card.

LIST: **$125**
OURS: **$109**

MDS

MDS, Inc. 207/772-5436

Quelo® 68000 Software Development Tools

Quelo Cross Assembler Packages are **Motorola compatible**. Each package includes a macro assembler, linker/locator, object librarian, utilities for producing **ROMable code**, extensive indexed typeset manuals and produces **S-records**, Intel hex, **extended TEK hex**, **UNIX COFF** and symbol cross references. **Portable source** written in "C" is available. It has been ported to a variety of mainframes and minis including **VAX, Sun, Apollo, Masscomp,** and **Charles River.** Native versions available for **Amiga** and **Atari ST.**

68020 Cross Assembler Package

Supports 68000, 68010, 68020, 68881and 68851.
For CP/M-68K and MS/PC-DOS, $ 750

68000/68010 Cross Assembler Package

For CP/M-80, -86, -68K and MS/PC-DOS, $539

68000/68010 Simulator

For MS/PC-DOS by Big Bang Software, Inc.

SIM68K is a cost-effective software simulator for debugging 68000/68010 code on an IBM-PC or compatible, $285.

Circle **no. 302** on reader service card.

Trademarks: CP/M, Digital Research; MS, Microsoft Corporation; Quelo, Quelo, Inc.

Double Your C Language Programming Productivity

Instant-C is an **incremental compiler** for C that makes every aspect of C programming as fast as possible. Load your existing C source code into *Instant-C.* Whenever you change your code with *Instant-C's* built-in full-screen editor, only the changed portions of your programs are recompiled. The fully automatic compilation process includes linking of your multiple modules, and therefore **takes no link time.** *Instant-C* compiles more than twice as fast as normal compilers. More importantly, compile time is proportional to how much code you change, not to the size of your entire program. With *Instant-C,* you will be running your program within a few seconds after editing a change.

Instant-C's over 350 precise diagnostic messages help you understand immediately how to fix the problem. *Instant-C* shows any errors on the editor screen, with the cursor placed exactly at the trouble spot.

Instant-C runs your program at compiled speeds with **full run-time checking** of pointer references and array indexes. Run-time checking catches problems as they occur, when they are easiest to fix and understand. This checking reduces the number of times you need to modify and test your program.

Instant-C has the **best testing and debugging** capabilities available. You have full access to the C language and to your program. At any time you can examine or modify variables (including locals), evaluate expressions or macros, and call functions interactively. You can examine and change your code, then resume execution. Use any number of conditional breakpoints. *Instant-C's* **visual debugging** shows your source code, highlighting the line being executed. *Instant-C* supports multiple screens, virtual screens, and non-standard graphics devices. *Instant-C's* **data monitor** stops execution when specified variables or memory areas are changed.

Instant-C is compatible with Lattice 2.x, 3.x and Microsoft 3.0, 4.0. Get *Instant-C's* superb debugging, run-time checking, and incremental compilation to **double your C programming productivity.**

Rational Systems, Inc.
P.O. Box 480
Natick, MA 01760
(617) 653-6194

Circle **no. 304** on reader service card.

HOURS
8:30 A.M. - 8:00 P.M. E.S.T.

800-421-8006

THE PROGRAMMER'S SHOP™
5-D Pond Park Road, Hingham, MA 02043
Mass.: 800-442-8070 or 617-826-7531

HOT GRAPHICS PACKAGE FOR C PROGRAMS.* $39.95

Everything you need to write dramatic graphics effects into your Eco-C88 C programs. Some of the features include:

- Support for EGA, CGA, and Z100
- Over 100 graphics and support functions, many of which are PLOT-10 compatible.
- Many low level support routines reside outside your small model code-data area
- Can write dots thru the BIOS (for compatibility) or to memory (for speed)
- Graphics function help from CED editor available
- World, pixel or turtle color graphics modes
- 47 standard fill patterns, 17 line dashing patterns, Hershey fonts, plus user defineable fill, dash and fonts
- Supports view areas, rotateable fonts, clipping, arbitrary fill areas, extensive error checking, examples, and user's manual.

A must for the graphics enthusiast and a bargain at only **$39.95**

***Requires Eco-C88 C Compiler.**

NEW POP-UP WINDOWS FOR YOUR C PROGRAMS.

This windowing library allows you to add pop up windows in your C programs quickly and easily. Use them for help windows, selection menus, error messages, special effects—anywhere you need an attention getter. Just some of the features include:

- CGA, EGA, and monochrome support
- Slow mode option for "flicker" displays
- Control any program that goes through the BIOS
- Use up to 255 windows
- No special window commands; use print f ()
- Resize and move windows
- Custom window titles and borders
- Can be used with ANSI device driver
- Most of window's code-data lies outside small model limits
- Use any of the IBM text or block characters
- User's manual and examples

The Windowing Library requires an IBM PC compatible BIOS and the Eco-C88 C compiler.

ONLY $29.95

HANDY LIBRARIAN MAKES LIFE EASIER.

Now you can combine your modules, functions, and subroutines into your own library for easy link commands. Fully compatible with ANY standard OBJ format files (not just Ecosoft's products). With the Ecosoft librarian, you can:

- Add, delete, and extract from a library
- Get table of contents or index of a library
- Combine libraries, control library page size, use switches for combinations, process complex library requests, use wildcards, and do library directives from command files.
- Complete with user's manual

A valuable addition for any programmer.

ONLY $29.95

THE FIRST PROFESSIONAL 'C' COMPILER FOR UNDER $60.

A C compiler with many ANSI enhancements at an unbelievably low price. The Eco-C88 C compiler has:

- Prototyping (the new type-checking enhancement)
- Enum and void data types
- Structure passing and assignment
- All operators and data types
- A standard library with more than 200 functions (many of which are System V compatible for greater code portability)
- CC and mini-make that all but automates the compile process
- 8087 support (we sense the 8087 at runtime — no dual libraries)
- ASM or OBJ output for use with MSDOS linker
- Tiered error messages — enable-disable lint-like error checking
- Fast compiles and executing code
- Expanded user's manual
- CED full-screen program editor

Everything you need at the unbelievable price of $59.95.

Eco-C88 C compiler requires an IBM PC, XT, or AT (or compatible) with 256K of memory, 2 disk drives and MSDOS 2.1 or later.

Orders only:
1-800-952-0472
Technical Information:
(317) 255-6476

NOT COPY PROTECTED.

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

ORDER FORM CLIP & MAIL TO: Ecosoft Inc., 6413 N. College Ave., Indianapolis, IN 46220

ITEM	PRICE	QTY	TOTAL
Flexi-Graph Graphics	$39.95		
Window Library	$29.95		
Eco-Lib Librarian	$29.95		
Eco-C88 C Compiler CED	$59.95		
SHIPPING			
TOTAL (IND. RES. ADD 5% TAX)			

PAYMENT: ☐ **VISA** ☐ **MC** ☐ **AE** ☐ **CHECK**

CARD # ______ EXPIR DATE ______

NAME ______

ADDRESS ______

CITY ______ STATE ______

ZIP ______ PHONE ______

Circle **no. 89** on reader service card.

ECOSOFT

Listing Nineteen

(Listing continued, text begins on page 130.)

```
    trap = (UCHAR *)( Linetrap + OLINE );

    while( ++line<=PGLEN  &&  line<=MAXLTRAP  &&  !*trap )
          trap += sizeof(LTRAP);

    return line - OLINE ;
}
```

End Listing Nineteen

Listing Twenty

```
/*----------------------------------------------------------
 * NRMAP.C: Routine to map strings of type char to strings
 *          of type CTYPE. The only externally accessable
 *          subroutine is:
 *                            void map( dest, src )
 *                            UCHAR  *src;
 *                            CTYPE  *dest;
 *
 *      Copyright (c) 1987  Allen I. Holub.
 *----------------------------------------------------------
 */

#include <stdio.h>
#include <ctype.h>
#include "nr.h"
#include "nrmap.h"

void    map( dest, src )
UCHAR   *src;
CTYPE   *dest;
{
    /* Map the input character array over to a CTYPE array
     * in order to get some room for attribute bits (ie.
     * bold, italics, overstruck, etc.). Set the attributes
     * as we process.
     *
     * Only printing characters can have attributes. Motion
     * is transmitted to text() as two bytes, the first
     * indicates the direction and the second is a count.
     * map puts the direction in the low byte and the count
     * in the high byte. If dest == 0 then the various
     * character attributes are set but no other processing
     * is done.
     */

    register  unsigned i, c ;

    while( i = *src )
    {
        ++src ;

        if( i==VMOVE  || i == HMOVE )
        {
            i = *src++ | MODE_BIT;

            i |= (i == VMOVE) ? VM_BIT : HM_BIT ;

            if( i & 0x80 )
                i |= 0x0f00;     /* Sign extend */
        }
        else if( i==CH_FONT )
        {
            Bold = Italics = Over = 0;  /* attributes off  */

            i = *src++ | (FONT_BIT | MODE_BIT);
        }
        else if( i==CH_ATTRIB )
        {
            if( *src )
            {
                switch( *src++ )
                {
                case BOLD:       Bold    = 1;      break;
                case ITALICS:    Italics = 1;      break;
                case OVER:       Over    = 1;      break;
                }
            }

            CLRWIDTH( i );
            continue;
        }
        else            /* Set the appropriate attribute bits */
        {
            c = i;

            switch(i)
            {
            case LITCHAR:      c=i=*src++ ;                    break;
            case SOFT_HYPHEN:  c=i=*src++ ; HYPHENATE(i);      break;
            case ZWIDTH:         i=*src++ ; c = -1;            break;
            case UP_SPACE:     c=i=' '    ; SETNOPAD(i);       break;
            }

            if((Num_under && isalnum(i)) || Cont_ul || Italics)
                   SET_UL( i );

            if( !WHITE(i) )              /* Don't boldface or */
            {                            /* overstrike spaces */
                if( Num_os    || Over )
                    SET_OS( i );

                if( Num_bold || Bold )
                    SET_BD( i );
            }

            if( 0 <= c  &&  c <= MAX_CHARS_IN_FONT )
                   SETWIDTH( i );
            else
                   CLRWIDTH( i );
        }

        if( dest )
               *dest++ = i ;
        else
               return;
    }

    if( Num_under ) --Num_under;
    if( Num_bold  ) --Num_bold ;
    if( Num_os    ) --Num_os   ;
    if( Cont_ul   ) --Cont_ul  ;

    *dest = 0;
}
```

End Listing Twenty

Listing Twenty-one

```
/*  NRMSC.C      Stuff that didn't fit anywhere else
 *
 *  (C) 1987, Allen I. Holub.
 */

#include <stdio.h>
#include <ctype.h>
#include "nr.h"

extern char     *skipto();

typedef struct
{
    unsigned adjusting :1;  /* adjustment enabled         */
    unsigned bold      :1;  /* boldface active            */
    unsigned fill      :1;  /* filling enabled            */
    unsigned italics   :1;  /* italics active             */
    unsigned over      :1;  /* overstrike active          */

    int     adjmode;       /* adjustment mode (.ad M)     */
    int     cmd;           /* current command character   */
    int     cont_ul;       /* lines to underline (.cu)    */
    int     curfont;       /* Current font (.f)           */
    int     esc;           /* current escape character    */
    int     indent;        /* current indent (.in)        */
    int     itrap;         /* current input line trap     */
    char    itrap_name[2]; /* name of the above           */
    int     lspace;         /* line spacing (.ls)         */
    int     nobreak;     /* current nobreak character     */
    int     nm_blanks;   /* line numbering stuff (.nm)    */
    int     nm_on;       /*        "                      */
    int     nm_mult;     /*        "                      */
    char    *nm_str;     /*        "                      */
    int     num_bold;    /* # of lines to boldface   (.bo) */
    int     num_center;  /* # of lines to center     (.ce) */
    int     num_under;   /* # of lines to underline  (.ul) */
    int     num_os;      /* # of lines to overstrike (.ul) */
    int     offset;      /* current page offset  (.po)    */
    char    *rmarg_str;  /* margin character              */
    char    *lmarg_str;  /*        "                      */
    int     tab;         /* tab expansion character       */
    int     leader;      /* leader expansion character    */
    int     linlen;      /* line length  (.ll)            */
    int     tempin;      /* temporary indent              */
    int     title_len;   /* length of 3-part title        */
    TSTOP   tabs;        /* previous tab stops            */
    int     *fillbuf;    /* fill buffer contents          */
}
ENVIORNMENT;

#define ESTACKSIZE      5

static ENVIORNMENT      Env_stack[ESTACKSIZE];
static int              Esp = ESTACKSIZE ;

/*----------------------------------------------------------------
 *              ENVIORNMENTS    (.ev command processing)
 */

push_env()
{
    ENVIORNMENT      *env;
    extern int       *saveq();

    if( Esp <= 0 )
    {
          err("Environment stack full\n");
          return;
    }

    env = &Env_stack[--Esp];

    env->adjmode          = Adjmode;
    env->adjusting        = Adjusting;
    env->bold             = Bold;
    env->cmd              = Cmd_chr;
    env->cont_ul          = Cont_ul;
    env->esc              = Esc;
    env->fill             = FILL;
    env->curfont          = CURFONT;
    env->fillbuf          = saveq();
    env->indent           = INDENT;
    env->italics          = Italics;
    env->itrap            = Itrap;
    env->itrap_name[0]    = Itrap_name[0];
    env->itrap_name[1]    = Itrap_name[1];
    env->lspace           = LSPACE;
```

(continued on page 100)

Listing Twenty-one

(Listing continued, text begins on page 130.)

```
        env->linlen          = LINLEN;
        env->nm_on           = Nm_on;
        env->nm_blanks       = Nm_blanks;
        env->nm_mult         = Nm_mult;
        env->nm_str          = Nm_str;
        env->num_bold        = Num_bold;
        env->num_center      = Num_center;
        env->num_under       = Num_under;
        env->num_os          = Num_os;
        env->nobreak         = Nobreak;
        env->offset          = OFFSET ;
        env->over            = Over;
        env->rmarg_str       = Rmarg_str;
        env->lmarg_str       = Lmarg_str;
        env->tab             = Tab;
        env->leader          = Leader;
        env->tempin          = Tempin;
        env->title_len       = Title_len;
        memcpy( env->tabs, Tabstop, NUMTABS );
        Num_under    =0;
        Num_bold     =0;
        Num_center   =0;
        Num_os       =0;
        Bold         =0;
        Over         =0;
        Italics      =0;
        Cont_ul      =0;
        Tempin       =0;
}

/*------------------------------------------------------------*/

pop_env()
{
        ENVIORNMENT     *env;

        if( Esp >= ESTACKSIZE )
        {
                err("Enviornment stack empty\n");
                return;
        }

        D(printf("Restoring from Env_stack[%d]\n", Esp));

        env = &Env_stack[Esp++];

        Adjmode          = env->adjmode;
        Adjusting        = env->adjusting;
        Bold             = env->bold;
        Cmd_chr          = env->cmd;
        Cont_ul          = env->cont_ul;
        Esc              = env->esc;
        FILL             = env->fill;
        restorq          ( env->fillbuf );
        INDENT           = env->indent;
        Italics          = env->italics;
        Itrap            = env->itrap;
        Itrap_name[0]    = env->itrap_name[0];
        Itrap_name[1]    = env->itrap_name[1];
        LINLEN           = env->linlen;
        LSPACE           = env->lspace;
        Nm_blanks        = env->nm_blanks;
        Nm_on            = env->nm_on;
        Nm_mult          = env->nm_mult;
        Nm_str           = env->nm_str;
        Num_bold         = env->num_bold;
        Num_center       = env->num_center;
        Num_under        = env->num_under;
        Num_os           = env->num_os ;
        Nobreak          = env->nobreak;
        OFFSET           = env->offset;
        Over             = env->over;
        Rmarg_str        = env->rmarg_str;
        Lmarg_str        = env->lmarg_str;
        Tab              = env->tab;
        Leader           = env->leader;
        Tempin           = env->tempin;
        Title_len        = env->title_len;
        memcpy( Tabstop, env->tabs, NUMTABS );

        if( CURFONT != env->curfont )
            chgfont( Fonts[CURFONT].name );

        CURFONT = env->curfont;
}

/*------------------------------------------------------------
 *      Tabs, Leaders, and Fields
 */

tabset( s )
char    *s;
{
        /* S is a string of comma or space delimited elements
         * of the form:             [+]Nt
         * where N is the position of the tab, t is the type
         * (L/C/R). The optional + means add N to the previous
         * tab stop value.
         */

        int prevtab = 0, tab = 0 ;

        for( ; *s ; prevtab = tab )
        {
            if( *s == '+' )
            {
                    s++;
                    tab = prevtab + uatoi( &s );
            }
            else
                    tab = uatoi( &s );

            if( tab < 1 || tab >= NUMTABS )
            {
                err("Tab stop must be in the range 1-%d\n",
                                                NUMTABS-1);
                return;
            }

            if( *s=='R' || *s=='C' || *s=='L' )
                    Tabstop[tab] = *s++ ;

            else if( *s==' ' || *s== ',' || !*s )
                    Tabstop[tab] = 'L';

            while( *s == ' '  || *s == ',')
                    s++;
        }
}

/*------------------------------------------------------------*/

tabclr()                                /*   Clear all tabs    */
{
        memset( Tabstop, 0, NUMTABS );
}

/*------------------------------------------------------------*/

tabprint()                              /*  Print tabs         */
{
        register int        i;

        for( i = OFFSET; --i >= 0 ; outc(' ') )
                ;

        for( i = 1; i <= LINLEN ; i++ )
                outc( Tabstop[i] ? Tabstop[i] : '.' );

        outc('\r');
        outc('\n');
}

/*------------------------------------------------------------
 * FONT support routines:
 */

findfont( fname )
int     fname;
{
        /* Search for a font in the font table ( Fonts[] ).
         * Return an index into the table or -1 if the
         * entry isn't there.
         */

        register int        i;
        register FONT       *fp;

        if( fname == 'R' )       /* Roman font is at Font[0] */
            return 0;

        if( isdigit( fname ) )
        {
                if( (i = fname - '0') > NUMFONTS-1 )
                    err( "%d: Illegal font number\n", i );

                else if( ! Fonts[i].name )
                    err( "Font number %d is undefined\n", i );

                else
                    return i;
        }
        else
        {
            fp = &Fonts[NUMFONTS-1];

            for( ; fp > Fonts ; --fp )
                if( fp->name == fname )
                    return( fp - Fonts );
        }

        return -1;
}

/*------------------------------------------------------------*/

chfont( nfont_name )
{
        /* Called from nrout.c to actually change fonts.
         * nfont and prevfont are indexes into the Fonts[]
         * table for the new and previous fonts. The CURFONT
         * number register holds the index for the current font.
         * Fonts[0] is the Roman font.
         */

        int             nfont;
        static int      prevfont = 0;    /* Roman */
        FONT            *p;

        if( nfont_name == PREVIOUS )
        {
            nfont    = prevfont;
            prevfont = 0;
        }
        else if( (nfont = findfont(nfont_name)) < 0 )
        {
            err( "Font <%c> unavailable, using (R)oman\n",
                                              nfont_name);
            nfont = 0;
        }
```

(continued on page 102)

Listing Twenty-one

(Listing continued, text begins on page 130.)

```
    if( CURFONT != nfont )
    {
        if( *Fonts[CURFONT].emac )
            expand_macro( Fonts[CURFONT].emac );

        p = &Fonts[nfont] ;

        Right_str = p->right      ;
        Left_str  = p->left       ;
        Hs_amt    = p->resolution ;

        if( *(p->smac) )

            expand_macro( p->smac );
    }

    prevfont = CURFONT;
    CURFONT  = nfont   ;
}
```

End Listing Twenty-one

Listing Twenty-two

```
/*------------------------------------------------------------
 * NROUT.C:     Output routines for nr.
 *
 * Copyright (c) 1987 Allen I. Holub.  All rights reserved.
 *------------------------------------------------------------
 */

#include <stdio.h>
#include <ctype.h>
#include <stdarg.h>
#include "nr.h"
#include "nrmap.h"
#include "nrtlen.h"

/*------------------------------------------------------------
 * isword(c)  evaluates to true if c can be in a word. Words
 *            are composed of all characters except padable
 *            space characters.
 */

#define isword(c) ( ! (WHITE(c) && PADDABLE(c)) )

/*----------------------------------------------------------*/

err( fmt )
char    *fmt;
{
    /* Print out an error message. If a macro is being
     * processed, the macro name is give, otherwise the
     * current input file name is given. This routine
     * works like printf() in all other respects.
     */

    va_list   args;
    va_start( args, fmt );

    fprintf( stderr, "\007ERROR(%s%s, line %d): ",
            Ismacro    ? "macro "  : "" ,
            *Ifilename ? Ifilename : "stdin", INLINES);

    vfprintf( stderr, fmt, args );
}

/*----------------------------------------------------------*/

ngaps( line, total_units, total_chars )
CTYPE   *line;
int     *total_units;
int     *total_chars;
{
    /* Returns the number of paddable space characters on
     * the line. Modifies *total_units to hold the number
     * of horizontal units occupied by non-space characters.
     * *total_chars is modified to hold the total number of
     * characters on the line.
     */

    register int utotal = 0;
    register int ngaps  = 0;
    CTYPE        *p;
    FONT         *ftab;

    for(p = line; *p ; ++p )
    {
        if( WHITE(*p) )
        {
            ++ngaps;
        }
        else if ISCHAR( *p )
        {
            utotal += CWIDTH( *p );

            D( printf( "ngaps(): %c ", *p )          );
            D( printf( "= %d units\n", CWIDTH(*p) ) );
        }
    }

    *total_chars = p - line ;
    *total_units = utotal;

    return ngaps;
}

/*----------------------------------------------------------*/

radjust( line, col )
CTYPE   *line ;
int     col   ;
{
    /* Do right adjustment. That is spread the words
     * as evenly as possible on the line so that the
     * rightmost character of the rightmost line is at
     * the indicated column (col). This is accomplished
     * by replacing all paddable space characters with
     * motion characters.
     */

    static int left = 0;
    int         num_units, num_chars, gaps, need ;
    int         space_between_characters;
    int         extra_space;

    gaps = ngaps( line, &num_units, &num_chars );

    if( gaps == 0 )
        return;
    need = (col * SPACE_SIZE) - num_units ;
    space_between_characters = need / gaps ;
    extra_space              = need % gaps ;

    D( outints( "radjust(), input line:", line )            );
    D( printf("Total units = %d, ",  num_units )            );
    D( printf("total chars = %d\n",  num_chars )            );
    D( printf("Padding to %d columns"    , col )            );
    D( printf("= %d units\n", col * SPACE_SIZE )            );
    D( printf("%d gaps spread over %d units\n", gaps,need));
    D( printf("%d units/gap, ", space_between_characters) );
    D( printf("%d extra units\n\n", extra_space)            );

    for(; gaps > 0 ; line++ )
    {
        if( WHITE(*line) )
        {
            --gaps;
            *line = MOTION( space_between_characters );

            if( left   || (gaps <= extra_space) )
            {
                if( --extra_space >= 0 )
                    ++(*line) ;
            }
        }
    }

    left = !left;
}

/*----------------------------------------------------------*/

justify( line, mode )
CTYPE   *line ;
int     mode  ;
{
    /* Do simple line adjustment on str (ie. do left,
     * right or center mode adjusting). If the
     * adjustment mode is BOTH then the routine radjust()
     * is called and we don't do anything here.
     */

    int     num_units, num_chars, need, gaps ;

    if( !*line || mode == LEFT )   /* Left adjustment   */
            return;                /* is no adjustment */

    if( mode == BOTH )
            return radjust( line, TLEN );

    gaps = ngaps( line, &num_units, &num_chars );
    need = U_TLEN - (num_units + (gaps * SPACE_SIZE));

    if( need <= 0 )
            return;

    memcpy( line + 1, line, (num_chars + 1) * sizeof(CTYPE));
    *line = MOTION( mode==CENTER ? need/2 : need );
}

/*----------------------------------------------------------*/

title( instr )
char    *instr ;
{
    /* Do a three part title:    /str1/str2/str3/
     * The character held in Page_ch is expanded to the
     * current page number. In the absence of an argument
     * do nothing.
     *
     * Titles are done using the normal output function.
     * The delimiters are replaced with normal spaces and
     * normal spaces are replaced with unpaddable spaces.
     * Then outbuf() is called with adjustment mode
     * turned on to print the line. Outbuf() will spread
     * the three parts of the title evenly on the line.
     */

    UCHAR           *s;
    int             delim, ndelim ;
    static CTYPE    dest[ MAXSTR ];
    int             i, fmt;
    UCHAR           pagenum[60];

    if( !(delim = *instr++) )
        return ;

    ndelim = 3;
    for( s = instr; *s ; s++ )
```

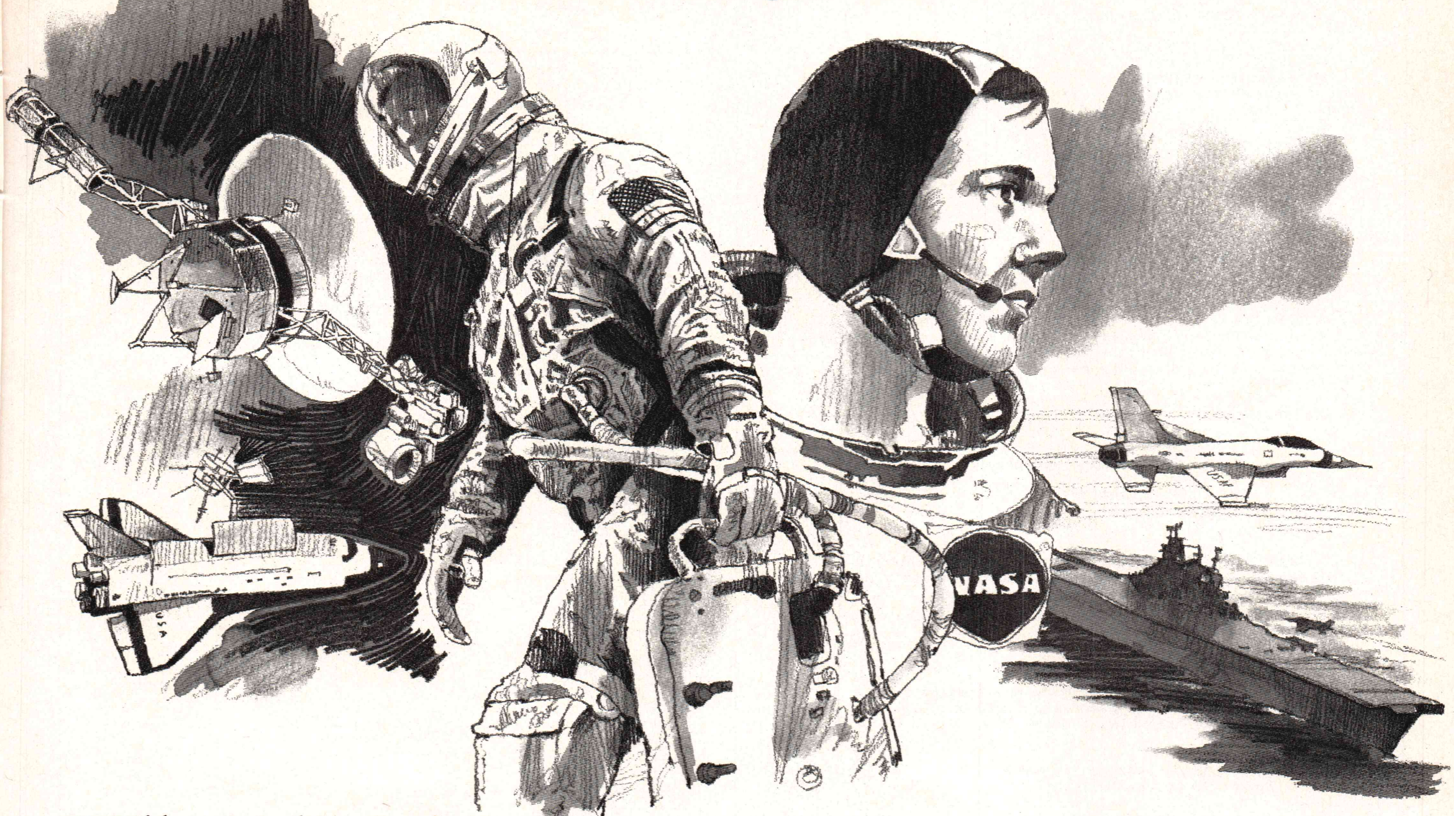
(continued on page 104)

Listing Twenty-two

(Listing continued, text begins on page 130.)

```
    {
        if( *s == ' ' )
        {
            *s = UP_SPACE;
        }
        else if( *s == Page_ch )
        {
            deletes( 1, s );

            i = nrtoi( "%",    &fmt          );
            itoascii ( pagenum, fmt, i       );
            s += inserts( pagenum, s, MAXSTR ) - 1;
        }
        else if( *s == delim )
        {
            if( --ndelim > 0 )
                *s = ' ';
            else
            {
                *s = '\0';
                break;
            }
        }
    }

    map      ( dest, instr      );   /* map to CTYPE array    */
    radjust  ( dest, Title_len  );   /* Spread the line       */

    pad      ( OFFSET           );   /* Output page offset,  */
    outs     ( dest             );   /*   title string,      */
    outchar  ( TO_CTYPE('\n')   );   /*   and a newline      */
}

/*---------------------------------------------------------*/

outbuf( line, addhyphen )
register CTYPE  *line;
{
    /* Output a line of text, adding indent, and page
     * offset. Add a \n at eol. Do line adjusting or
     * centering as required. If adhyphen is true, a hyphen
     * is added immediatly after the line is printed.
     *
     * The line array must be MAXSTR characters long.
     *
     * This routine is called by dofill when filling is on
     * and is called directly by text when it is not. If
     * Nospace is enabled (by a .ns command) then lines
     * consisting of single newline characters will not
     * be printed.
     *
     * If *line is null then we are printing a blank line.
     * In this situation line numbering (.nm) is not done
     * and only one blank line will be printed, regardless
     * of the line spacing (.ls). If there is not left or
     * right margin string defined, no offset is printed.
     */

    register int        i ;
    char                numbuf[32] ;

    D( outints( "outbuf", line );  )

    if( !*line  &&  Nospace )
            return;

    Nospace = 0;      /* Re-enable spacing on a non-blank */
                      /* line.                            */
                      /* Then output the page offset and  */
                      /* the left margin string           */

    if( !Isdiv && (*line  || *Lmarg_str || *Rmarg_str) )
            pad( OFFSET );

    if( *Lmarg_str )
            outchs( Lmarg_str );

    if( Nm_on && (Nm_blanks || *line) )
    {
        /* Number the line if necessary
        */

        if( LINE % Nm_mult )
                pad( strlen(Nm_str) + 3 );
        else
        {
                sprintf(numbuf, "%3d%s", LINE, Nm_str );
                outchs( numbuf );
        }
        LINE++;
    }

    if( *line )         /* followed by the line if there  */
    {
            pad( INDENT + Tempin );

            /* Now, either center or adjust the line as
             * required. Note: Num_center (set by the .ce
             * command) is different from a centering
             * adjustment mode (ie. the former applies to
             * input lines, the latter to output lines.
             * Centering and adjusting are mutually exclusive.
             */

            if( addhyphen )
            {
                /* This kludge tricks justify() into
                 * thinking that the line is shorter than
                 * it really is when we add a hyphen.
                 */
                    Tempin++;
            }

            if ( Num_center )
            {
                    justify( line, CENTER );
                    --Num_center;
            }
            else if( Adjusting )
                    justify( line, Adjmode );

            if( addhyphen )
                    --Tempin;
    }
}
pad(i)
register int    i;
{
        /* Print i spaces using outchar()
        */

        while( --i >= 0 )
                outchar( TO_CTYPE(' ') );
}

/*-------------------------------------------------------*/

outc( c )
int     c;
{
    /* Lowest level output function. Handles control
     * character suppression. "Ispage" tests to see if a
     * page is in the list specified with a -o command line
     * switch. Note that direct calls to outc don't update
     * any global variables. outchar() is the normal output
     * function and should be used except in wierd
     * situations (ie. the .ou command). Note that \n is not
     * translated into \r\n by outc (it is so translated by
     * outchar).
     *
     * Note that outc() processes normal char's, not CTYPE's.
     *
     * A minor problem here is diversions. We are writing
     * in untranslated mode so that control sequences can
     * get to the printer unmolested. With diversions though,
     * we need to translate character by hand (strip off \r)
     * so that when they are read back (via mgetc) they
     * won't be re-translated on output.
     */

    c &= 0xff ;

    if( Isdiv )                  /* If we're processing a   */
    {                            /* diversion, put the text */
        if( c != '\r' )          /* there.                  */
            mputc( c, Ofile );
    }
    else if( ispage( PAGE ) )
    {
        /* If this is a printing page as per the -o flag
         * print the character
         */

        if( No_cntl &&   (c < ' ' || c > 0x7f)  )
        {
            printf( "<%02x>", c );
            if( c == '\n' )
                    printf( "\n" );
        }
        else
        {
            /* It's virtually impossible to get untranslated
             * standard output out of the Microsoft compiler.
             * so do it here with a direct DOS call.
             */

            bdos( 2, c & 0xff, 0 );   /* putc( c, stdout ) */
        }
    }
}

/*-------------------------------------------------------*/

outchar( big_c )
CTYPE   big_c;
{
    /*  Medium level character output function. Translates
     *  attributes into strings (ie. for bold chars etc.).
     *  Updates the line and page numbers when necessary.
     *  Springs traps as required.
     *  Handles the various motion escape sequences \u \d etc.
     *  Returns the width on the output line of c. This will
     *  be 0 if c is a motion character, a change font, a
     *  newline, etc. It will be 1 otherwise.
     *
     *  Note that outchar() is passed CTYPE's, not normal
     *  characters.
     */

    register unsigned int rval, c ;
    static   int            lastch = 0;
    int                     width;

    rval = 0 ;

    if( ISMOTION(big_c) )
    {
        do_ul   ( (CTYPE)0 );  /* All attributes off first */
        do_bold( (CTYPE)0 );
```

(continued on page 106)

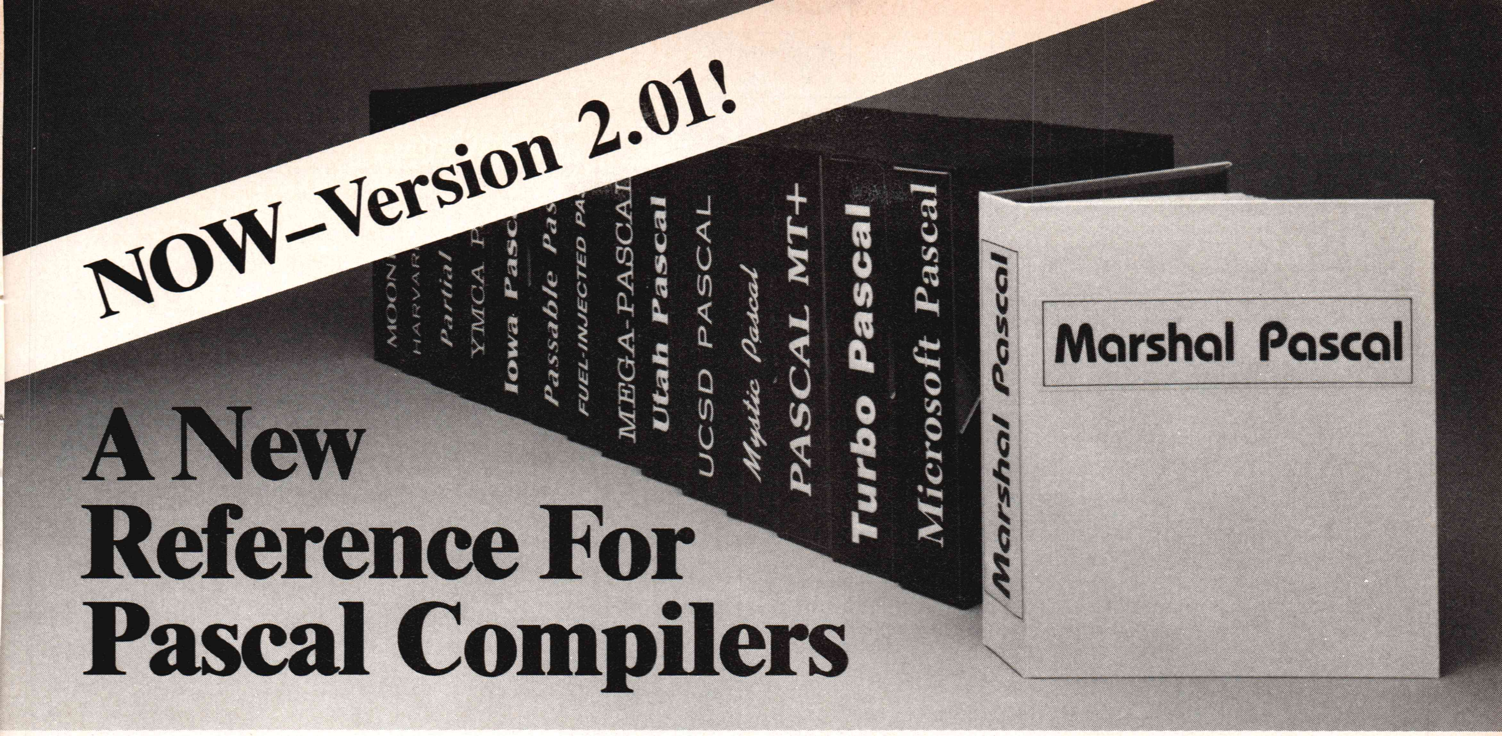

NASA

Listing Twenty-two

(Listing continued, text begins on page 120.)

```
        do_over( (CTYPE)0 );
        motion( MVAL(big_c), big_c );
    }
    else if( ISFONT(big_c) )
    {
        do_ul   ( (CTYPE)0 );
        do_bold( (CTYPE)0 );
        do_over( (CTYPE)0 );
        chfont( FVAL(big_c) ); /* then change fonts        */
    }
    else
    {                           /* ... or print a character:    */
        do_ul    (big_c);  /* Underline  character if reqd.*/
    TEXTLEN = outs( line );      /* Finally output the line */

    if( addhyphen )
    {
            TEXTLEN++;
            outchar( TO_CTYPE('-') );
    }

    /* Print the right margin character if needed. The
     * first call to pad() (in the if) prints the left
     * margin padding only if this is a blank line (normaly
     * we don't want to print any padding on blank lines).
     * The call to pad() gets us to the right margin.
     */

    if( *Rmarg_str )
    {
            if( !*line )
                    pad( INDENT + Tempin );

            pad( TLEN - TEXTLEN );
            outchs(Rmarg_str);
    }

    /* Reset Temporary indent. We couldn't do it earlier
     * because it's used in the TLEN macro. We don't want
     * to reset it if the line is blank though.
     */

    if( *line )
            Tempin = 0;

    /* Output at least 1 but as many newlines as are
     * required by the .ls N command. Only one line is
     * output if we are doing a blank line.
     */

    for( i = (*line ? LSPACE : 1);  --i >= 0;  )
        outchar( TO_CTYPE('\n') );
}

/*--------------------------------------------------------*/

outs( line )
CTYPE   *line;
{
        /* Output an integer string using outchar().
         * Returns amount of space occupied on the output
         * line by p. This will not include motion, font
         * changes, etc.
         */

        register rval = 0 ;

        D( outints( "outs", line);  )

        while(  *line  )
                rval += outchar( *line++ ) ;

        return rval;
}

#ifdef DEBUG /*-------------------------------------------*/

outints( str, line )
char    *str;
CTYPE   *line;
{
        /* Output a CTYPE string using putchar, putting
         * "str" in front of it.
         */

        printf("%s <", str );

        for(; *line ; putchar( *line++ & 0xff ) )
                ;

        printf(">\n");
}

#endif /*-------------------------------------------------*/

outchs( str )
char    *str;
{
        /* Output a character string using outchar()      */

        while( *str )
                outchar( TO_CTYPE( *str++ ) );
}

/*--------------------------------------------------------*/

ots( str )
char    *str;
{
        /* Output a character string using outc.  Note that
         * \n will not be translated into \r\n and no global
         * vars will be updated.
         */

        while( *str )
                outc( *str++ );
}

/*--------------------------------------------------------*/
        do_bold (big_c);  /* Boldface   character if reqd.*/
        do_over (big_c);  /* Overstrike character if reqd.*/
        c = CHAR(big_c);  /* Translate to normal char     */

        if( c == '\n')
        {
            nextline( lastch == '\n' );
        }
        else
        {
            width = CWIDTH( big_c );

            if( width > 1 )      /* In a proportional font */
                motion( width/2, (CTYPE)(MODE_BIT|HM_BIT));

            outc( c );           /* Print char and advance */
                                 /* 1 HMI unit             */

            if( width > 1 )
                motion( (width/2) - 1,
                        (CTYPE)(MODE_BIT|HM_BIT));

            if( !HASWIDTH(big_c) )
                motion( -Fonts[CURFONT].widths[c],
                        (CTYPE)( MODE_BIT | HM_BIT ));

            rval = 1;
        }

        lastch = c;
    }

    return rval;
}

/*--------------------------------------------------------*/

motion( count, how )
int     count;
CTYPE   how;
{
    /*  Take care of motion. \u \d \v etc.  */

    register int        negative = 0;

    if( count < 0 )
    {
            negative = 1;
            count = -count;
    }

    while( --count >= 0 )
    {
            if( HORIZONTAL(how) )
                    ots( negative ? Left_str : Right_str );
            else
                    ots( negative ? Up_str   : Dn_str );
    }
}

/*--------------------------------------------------------*/

nextline( harder )
{
    /*  -  Adjust the line and page numbers or diversion
     *     height as appropriate.
     *  -  Spring a line trap if one is present. A line
     *     trap is expanded when a "line of text is output
     *     whose vertical size reaches or sweeps past the
     *     trap position." In other words, a line trap at
     *     line 10 will be triggered immediatly after line
     *     10 is printed. Trap 0 is used to spring a trap
     *     at the top of the page; it is done in outchar()
     *     before the first character on a page is printed
     *     (we can't do it at the end of the current page
     *     because it wouldn't be sprung on the first page).
     *  - Adjust various number registers as appropriate.
     *  - Process the -s command line switch if one was
     *     given.
     *  - Translate \n into \r\n or do Wordstar mode if
     *     necessary.
     */

    extern int  Stop;            /* Declared in nr.c      */

    if( !Wordstar )              /* Not in wordstar mode */
    {
            outc( '\r' );
            outc( '\n' );
    }
    else if( harder )            /* In wordstar mode:     */
    {
            outc( '\r' );        /* Saw two successive    */
            outc( '\n' );        /* newlines in           */
            outc( '\r' );        /* outchar().            */
            outc( '\n' );
    }
    else if( Wordstar == 1 )     /* Use Wordstar soft     */
    {                            /* carriage return for   */
```

(continued on page 108)



	Ackerman		Sieve		I/O		Gauss-Seidel		Floating Point	
Marshal Pascal	**11.9**	**3.5K**	**4.8**	**2.4K**	**1.8**	**5.1K**	**4.9**	**11.5K**	**10.5**	**3.2K**
IBM Pascal	12.4	34.7K	11.7	27K	2.5	24.5K	7.6	34.6K	10.5	21K
Turbo Pascal	22.7	11.6K	14.2	11.5K	2.2	12.5K	4.7	13.5K	28	11.4K
Oregon Pascal-2	18.2	13.9K	7.2	11.7K	2.5	22K	11.8	37.6K	139.7	26.8K
Microsoft C 4.0	15.9	9.3K	5.8	6.5K	1.9	8.9K	6.0	23.6K	33	19.6K
	Sec.	Code Size	Sec.	Code Size	Min.	Code Size	Sec.	Code Size	Sec.	Code Size

Unparalleled Speed and Power

Marshal Pascal™ is the most highly code-optimized Pascal compiler for PCs. Period. In fact, Marshal Pascal produces code so fast and compact that even the most efficient C compilers fall behind in performing many operations. Also, Marshal Pascal is an ISO implementation thus offering portability to other computer environments. You may address as much memory as your operating system allows and a variety of memory models are supported. Among the useful extensions included are: separate compilation of modules (both Modula-2 and Pascal forms), structured constants and structured function values, variable-length string types and procedural parameters.

Turbo Pascal® Translator

Our translator brings your Turbo Pascal software over to an ISO/Marshal-readable format. You can watch your present Turbo programs run in a fraction of their former time and code space!

8087-80287 Support

Marshal Pascal supports the Intel 8087-80287® math processors *inline*. If you don't have the math chip, then '87-287 code simulation is a provided option.

Microsoft Linkability

Marshal Pascal's true relocatable linker allows you access to the Microsoft family of languages and assemblers. A flexible object code librarian and powerful overlay capabilities are also included.

Powerful Compile Options

Marshal Pascal gives you a number of compile options, including an optimization by-pass for speedier compiles, I/O "fine-tuning", constant folds and a syntax evaluator just to name a few. A wealth of compile-time checks permits you to find the more subtle logic errors, reducing debugging time enormously.

Efficient Large Heap Model for AI Applications

Marshal Pascal gives users efficient dynamic allocation for symbolic processing in AI applications. Marshal Pascal also allows functions to return arbitrary structured types. This can be used to combine the efficiency of Pascal with the powerful functional programming style enjoyed by such languages as LISP.

The Price? $189, *includes everything.*

Supports PC-DOS®, MS-DOS®, CP/M-86®, Concurrent DOS®, and *soon* Xenix 286/386®!

To order your copy of **Marshal Pascal,** call: **(800) 826-2222**
In California: **(415) 947-1000**

Marshal Language Systems

1136 Saranap Ave., Ste. P, POB 2010, Walnut Creek, CA 94595

Listing Twenty-two

(Listing continued, text begins on page 130.)

```
        outc( 0x8d );          /* newlines           */
        outc( '\n' );
    }
    else                       /* Replace newline with */
        outc( ' ' );           /* a space character    */


    if( Isdiv )       /* Adjust diversion height and width  */
    {                 /* and spring trap if appropriate     */
        if( Divwidth < TEXTLEN )
            Divwidth = TEXTLEN ;

        if( Divtrap == VERT )
            do_divtrap();

        ++VERT ;     /* increment diversion height */
    }
    else
    {
        /* Spring any line traps for the current line and
         * then adjust the distance to the next trap. Note
         * that the line number has to be incremented as
         * part of the do_linetrap() call or else the trap
         * for the current line will be sprung recursively.
         */

        do_linetrap( OLINE++ );

        TOTRAP = distance();

        if( OLINE > PGLEN )
        {
            if( Stop  &&  !(PAGE % Stop) )
            {
                /* Process the -s command line switch */

                fprintf(stderr,"\nHit any key to continue.");
                getchar();
            }

            OLINE = 1 ;         /* Adjust line and page #    */
            ++PAGE;

            do_linetrap(0);   /* Spring top of page trap */

            TOTRAP = distance();
        }
    }
}

/*----------------------------------------------------------*/

go_up( amt )
{
    /* Called from sp(), in nrprocs.c, to handle negative
     * spacing. Amt is a negative number.
     */

    if( Isdiv )
        VERT = max( VERT + amt, 0 );
    else
    {
        OLINE  = max( OLINE + amt, 1 );
        TOTRAP = distance();
    }

    motion( Vs_amt * amt, (CTYPE)( MODE_BIT | VM_BIT ));
}

/*----------------------------------------------------------*/

do_ul( c )
CTYPE    c;
{
    /*  Take care of underlining c but don't actually print
     *  c. If Ul_on is defined toggle underline mode at the
     *  printer at the appropriate times. If Ul_on is not
     *  defined then just print a "_\b".
     */

    static   int amunder  = 0 ;

    if( Plain )
        return;

    if( IS_UL(c) )
    {
        if( *Ul_on )
        {
            if( !amunder )
            {
                amunder = 1;     /* Turn on underlining   */
                ots( Ul_on );
            }
        }
        else
            ots( "_\b" );
    }
    else
    {
        if( *Ul_off && amunder )
            ots( Ul_off );

        amunder = 0;              /* Turn off underlining */
    }
}

/*----------------------------------------------------------*/

do_bold(c)
CTYPE    c;
{
    /* Same as above but do boldface instead of underline
     */

    static   int ambold    = 0 ;

    if( Plain )
        return;

    if( IS_BD(c) )
    {
        if( *Bd_on )
        {
            if( !ambold )
            {
                ambold = 1;
                ots( Bd_on );
            }
        }
        else
        {
            outc( (int) c );
            outc( '\b');
        }
    }
    else
    {
        if( *Bd_off && ambold )
            ots( Bd_off );

        ambold = 0;          /* Turn off boldface */
    }
}

/*----------------------------------------------------------*/

do_over(c)
CTYPE    c;
{
    /*  Same as above but do overstrike.
     */

    static   int am_os    = 0 ;

    if( Plain )
        return;

    if( IS_OS(c) )
    {
        if( *Os_on )
        {
            if( !am_os )
            {
                am_os = 1;
                ots( Os_on );
            }
        }
        else
            ots("-\b");
    }
    else
    {
        if( *Os_off && am_os )
            ots( Os_off );

        am_os = 0;          /* Turn off overstrike */
    }
}
```

End Listing Twenty-two

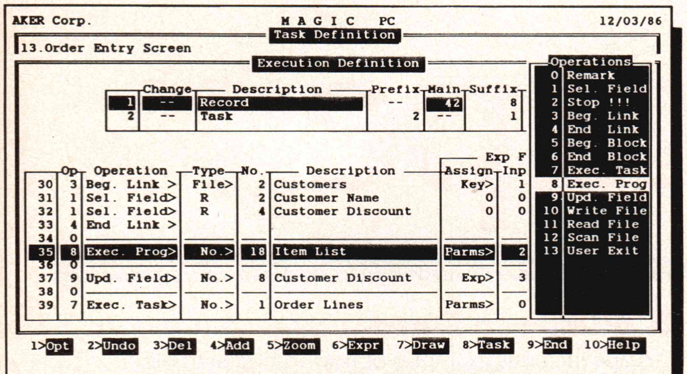
Listing Twenty-three

```
#include <stdio.h>
#include "nr.h"

/*----------------------------------------------------------
 * NRTAB.C      --      Tab-processing stuff for NR
 *
 * (C) 1987,  Allen I. Holub, All rights reserved.
 *
 * Tab processing is rather primitive. In particular, it
 * assumes that we're using a monospaced font.
 */

int width( c, advance )
int c, *advance;
{
    /* Return the amount of space taken by the character
     * on the output line. Modifiy "advance" to be the
     * amount of space required to skip past it.
     */

    switch( c )
    {
    case VMOVE:         *advance = 2;   return 0;
    case HMOVE:         *advance = 2;   return 0;
    case CH_FONT:       *advance = 2;   return 0;
    case CH_ATTRIB:     *advance = 2;   return 0;
    case ZWIDTH:        *advance = 2;   return 0;
    case LITCHAR:       *advance = 2;   return 1;
    case SOFT_HYPHEN:   *advance = 1;   return 0;
    }

    *advance = 1;       /* UP_SPACE and default case */
    return     1;
}
```

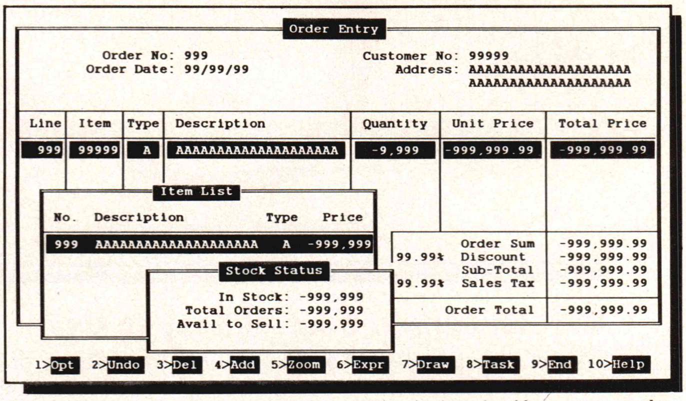
(continued on page 112)

Announcing Magic PC – the first breakthrough for database applications developers in over 20 years:

Now you can develop professional applications 1000% faster than your 4GL or DBMS, totally free from programming, commands and syntax!



A Magic PC program looks as simple as this. To design an application you quickly fill-in menu-driven decision tables **without having to write a single line of code.** For example, just by highlighting the Execute Program operation on this screen and also highlighting the Item List program in the Program Menu, you tell Magic PC to pop-up the Item List window shown in the adjacent screen, when the end-user hits the Zoom key.



Magic PC gives your end-user the power to harness and retrieve data instantly, **without any commands or syntax** because at runtime you already have built-in options to Add, Delete, Modify, Query and get on-the-spot ad-hoc information simply by highlighting selections from menus. Data validation, security and error-checking are done automatically for you by Magic PC without programming.

Who needs another DBMS?

At last, Magic PC gives you the ultimate applications design tool, far ahead of 4GL's, DBMS and Application Generators.

Magic PC breaks through the language barrier with the revolutionary Un-Language concept:

NO PROGRAMMING, COMMANDS OR SYNTAX!

Free yourself from your programming language

Magic PC makes you, the professional, completely free from the drudgery of procedural programming. No more cryptic commands, syntax or unforgiving procedural structures, because Magic PC does all the programming automatically. There's your competitive edge. The rest is up to you . . .

The Professional Choice

Already an international success, Magic PC is a profit maker and career booster for DP Consultants, System Integrators, VARs, MIS professionals, System Analysts, Programmer Analysts and Software Engineers. If you design PC applications professionally, you can't afford not to Un-Language now.

IBM France: "IBM encourages this introduction and can not help but salute such evolution . . ."

Israeli Air Force: "We were convinced that it was not possible to have a design tool powerful enough to implement real-life applications without a programming language. Magic PC changed our mind . . ."

Jeff Duntemann, PC Tech Journal: "It's probably the best integrated database applications and screen generator that I have ever seen . . .very smooth system, and smoothness comes at a premium these days . . ."

The Magic PC Secret

You're so much more productive with Magic PC because there is **absolutely no programming** to slow you down. You design a Magic PC application by simply filling-in the **Data Dictionary Tables** (Files, Fields, Keys) and the **Task Description Tables** (Operations and Expressions).

Only 13 design **Operations** harness the power of Magic PC. Operations are specific enough to eliminate the need for tiresome syntax, yet elastic enough to produce robust custom applications. Use the Operations to describe **what** you want and Magic PC makes it happen. It's that simple.

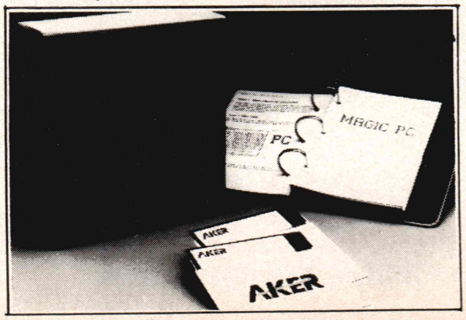
Make Task nesting power available with a single **Execute Task** Operation. This powerful instruction triggers Magic PC to execute and display additional tasks or even external applications through **Window Zooms.** The 3-dimensional effect of Window Zooming lets you probe deep into your application through nested windows and manipulate the data underneath.

You describe a Magic PC Task or Program (composite Tasks) by filling your system analysis flow into the Task Description Tables. Choose the participating Data View, and Magic PC executes your desired Operations. You interface with the Tables by highlighting your selections from pop-up menu-driven windows. There's nothing to edit except your headings.

You're not confined to any particular design sequence as you are with most procedural languages. You can enter and change any Table spontaneously, on the fly, as ideas come to mind and Magic PC automatically maintains the application integrity.

A **Magic Inference Engine** automatically orchestrates your Task Description Tables into a single file of internal **Knowledge Base Rules** for optimum, bug-free performance. Knowledge Base Rules are executed by the **Magic Run** engine for stand-alone runtime operation, or by the **Magic Lan** engine for unrestricted Novell network sharing. You're free to design the Knowledge Base without worrying about the internal structure.

Discover fast, language-free programming at no risk for only $19.95



See for yourself how fast you can program language-free applications with our low-cost limited offer.

You'll get the full Magic PC software unprotected and limited to 100 records and 450 page documentation complete with a **free** Order Entry sample application. You'll also get our **free** telephone support for 90 days!

And your $19.95 will be credited towards the full $695 Magic PC purchase price. Even if you don't buy Magic PC right away, keep your $19.95 Magic PC Trial as your application prototyping tool at this bargain price.

Our No-Risk Guarantee!

You have our no-risk 30-day money-back guarantee: if you're not completely satisfied for any reason, even Magic PC Trial for $19.95, send it back for a refund.

Order now while supply lasts

Call this toll free number now with your Visa, MasterCard or American Express for immediate delivery, or send the Order Coupon below today to Aker.

1-800-345-MAGIC
in CA call 714-250-1718

VISA MasterCard AMERICAN EXPRESS

Yes, please rush me:

☐ Magic PC Trial	$ 19.95
☐ Magic PC	$695.00
Add shipping	$ 5.00
In CA 6% tax	$_____
Prices valid in US only. Total	$_____

Ship to:__________
Address:__________
City/ST/Zip:__________
Phone:__________

AKER

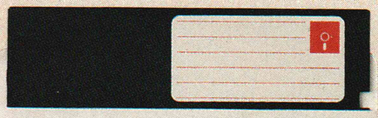
Aker Corp. 18007 Skypark Circle B2, Irvine, CA 92714
(714) 250-1718, Elec. Mail Dialcom 41:AKR 001 Telex 4931184 AKR UI OEM and VAR inquiries are welcome.

Min. requirements PC DOS 2.0, IBM PC or 100% compatible with 512K and hard disk.

Circle **no. 369** on reader service card.

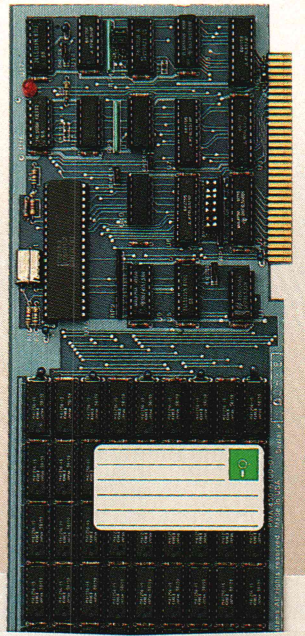
Think fast! Pick the better fit...

Our 5th Year Bonus!
Mention this ad when ordering
and get your choice of a V 20-8
(replaces 8088) or $20 off on
your Battery Backup purchase!



FLOPPY DISK.

- Fills time between coffee breaks
- Makes a hard disk seem *fast*
- Your computer appears busy (even if you aren't!)
- Wears out moving parts



SEMIDISK Disk Emulator.

- Gets that job done NOW
- Makes a hard disk seem *slow*
- Maximizes your productivity with anything from databases to compilers
- Totally silent operation

...for YOUR demanding tasks.

SURPRISE! *Neither* is memory mapped, so they don't affect your precious Main Memory. *Both* retain data indefinitely - even with the computer turned off.

THE SEMIDISK SOLUTION. You could invest in a series of "upgrades" that turn out to be expensive band-aids without solving your real problem. Even those "Accelerator" and "Turbo" boards do little to speed up disk-bound computers. If your applications spend too much time reading and writing to disk (and whose don't?), you won't want to settle for anything less than a SemiDisk disk emulator. The SemiDisk comes in 512K and 2Mb capacity. More boards may be added to make up to an 8 Megabyte SemiDrive!

SPEED THAT'S COMPATIBLE. PC, XT or AT, if you need speed, the SemiDisk has it. How fast? Recent benchmarks show the SemiDisk is from 2 to 5 times faster than hard disks, and from 25% faster (writing) to several times faster (random reads) than VDISK and other RAMdisk software that gobble up your main memory.

MEMORY THAT'S STORAGE. Using our small external power supply, with battery backup, your data remains intact through your longest vacation or even a seven-hour power failure!

CELEBRATE WITH US! Now, SemiDisk celebrates its fifth birthday with a special offer for IBM-PC owners. Buy a SemiDisk now and we'll include an 8 MHz V-20 microprocessor (replaces the 8088) to make your new SemiDisk run even faster. Don't need the V-20? We'll take $20 off the price of your Battery Backup Unit!

	512K	2Mbyte
IBM, PC, XT, AT	$495	$ 795
Epson QX-10	$495	$ 995
S-100 SemiDisk II	$795	$1295
S-100 SemiDisk I	$299	-------------
TRS-80 II, 12, 16	$495	$ 995
Battery Backup	$130	$ 130

Someday you'll get a SemiDisk.
Until then, you'll just have to...wait.

SemiDisk

MasterCard VISA

SemiDisk Systems, Inc., 11080 S.W. Allen Blvd., Beaverton, Oregon 97005 (503) 626-3104

Circle **no. 85** on reader service card.

Listing Twenty-three

(Listing continued, text begins on page 130.)

```
/*----------------------------------------------------------*/

dist_to_tab( col )
{
    /* Col is the current column position. Return the
     * distance the the next tab set in the Tabstop array.
     * That is, the next tab will be at Tabstop[col + rval].
     * A tab at the current column position is ignored.
     */

    register int        ocol;

    for( ocol = col++; !Tabstop[col] && col < LINLEN; col++)
            ;

    return (col < LINLEN)  ?  (col - ocol)  :  0 ;
}

/*----------------------------------------------------------*/

field_width( p )
UCHAR   *p;
{
    /* Return the distance beween p and either end of line
     * or a tab character (\t) or a leader character (SOH).
     */

    register int     count = 0;
    int              advance;

    if( !*p++ )
            return 0;

    while( *p  &&  *p != '\t'  &&  *p != SOH  )
    {
            count += width(*p, &advance);
            p     += advance;
    }

    return count ;
}

/*----------------------------------------------------------*/

dotab( str )
UCHAR   *str;
{
    /* Expand the tab (^I) and leader (^A - SOH)
     * characters in str to the proper number of tab or
     * leader characters.
     *
     * Three types of tabs are recognized. (L)eft justifying
     * tabs will print the character following the \t at the
     * tabstop. Right and Centering tabs both use a field
     * width (ie. the number of character following the \t
     * up to, but not including, a following \t or end of
     * line). A centering tab will cause this "field" to be
     * centered on the tabstop. A right adjusting tab will
     * cause the rightmost character in the field to rest
     * immediately to the left of the tabstop.
     */

    int      w;             /* Current field width            */
    int      d;             /* Distance to next tabstop       */
    int      col ;          /* Current output column          */
    UCHAR    *p;            /* Current character              */
    UCHAR    *startstr;     /* Original beginning of string   */
    int      advance;       /* amount to advance past char    */

    static  UCHAR   buf[ MAXSTR ];
    col      = 1   ;
    startstr = str ;

    /* Copy str into buf with strncpy() and then copy it
     * back expanding tabs an leaders.
     */

    strncpy(buf, str, MAXSTR);

    for( p = buf; *p ;)
    {
        if(  !(*p == '\t' || *p == SOH)  )
        {
            if( str-startstr >= MAXSTR-1 )  /* out of space */
                break;

            col += width(*p, &advance);

            while( --advance >= 0 )
                    *str++ = *p++;

            continue;
        }

        if( !(d = dist_to_tab(col)) )
            break;

        w = field_width( p  );

        /* Convert d to the number of spaces to print
         * to get to the specified tab stop. If there
         * are no characters between the current
         * \t and the next \t then just move to the next
         * tab.
         */
```

```
        switch( Tabstop[ col + d ] )
        {
        case 'R':  d -= w    ;   break ;
        case 'C':  d -= w/2  ;   break ;
        }

        while( --d >= 0 )
        {
            if( str-startstr >= MAXSTR )
                    goto exit;

            *str++ = ( *p == '\t' ? Tab : Leader );
            col++;
        }

        p++;
    }

exit:
    *str = 0;
}
```

End Listing Twenty-three

Listing Twenty-four

```
#include <stdio.h>
#include <ctype.h>
#include "nr.h"
#include "nrmap.h"
#include "nrtlen.h"

/*----------------------------------------------------------
 *  NRTEXT.C:   Text processing portion of nr
 *
 *  Copyright (c) 1985 Allen I. Holub.  All rights reserved.
 *----------------------------------------------------------
 */

typedef CTYPE   QUEUE;  /* Dummy typedef for queue routines */


/*----------------------------------------------------------*/


extern QUEUE *makequeue() ;
extern CTYPE *show_next() ;
extern void  map(CTYPE*, char*);

/*----------------------------------------------------------
 * Globals:
 *
 * blankline:   Used in outbuf() calls when we need to print
 *              a blank line.
 * Input_queue: Input queue used for line filling. Words are
 *              enqueued until an entire line is in the
 *              queue, the the queue's contents are printed.
 * Last_queued: Most recently enqueued character.
 * Owidth:      Space occupied on the output line by the
 *              characters now in the queue, in units
 *              (modified by putq() and used in several
 *              places).
 */

#define QSIZE   (MAXSTR * 2)

static  CTYPE   blankline = NULL;

static  QUEUE   *Input_queue ;
static  CTYPE   Last_queued  ;
static  int     Owidth     = 0;

/*----------------------------------------------------------*/

void    text( str )
UCHAR   *str;
{
    /*Highest-level text processing routine, called
     * from process().
     */

    CTYPE           line [MAXSTR];
    static int  been_called = 0;

    D( printf("text(): working on <%s>\n", str); )

    if( Inhibit )          /* Input has been inhibited by an */
          return;          /* .if or .ie command             */

    if( !been_called )
    {
        /* If this is the first time we've been called,
         * spring the top of page macro for the first page.
         * (all other top of page macros are sprung
         * imediately after the bottom line of the previous
         * page is printed.
         */

        been_called = 1;
        do_lInetrap( 0 );
        TOTRAP = distance();
    }

    if( TLEN >= MAXSTR )
    {
            err("Output line too long\n");
    }
    else if( !*str )
```

(continued on next page)

Listing Twenty-four

(Listing continued, text begins on page 130.)

```
{
    /* A blank line always causes a break use `sp 1 if
     * you want blank lines without a buffer flush.
     * Note that outbuf won't print the blank line
     * itself unless spacing is enabled.
     */

    brk();                      /* Break                  */
    outbuf( &blankline, 0 );    /* print the blank line */
}
else
{
        map( line, str ) ;

        if( FILL )
                dofill( line );
        else
                outbuf( line, 0 );
}


if( Itrap > 0 )                        /* Input line trap */
{
        if( --Itrap == 0 )
                expand_macro( Itrap_name );
    }
}

/*--------------------------------------------------------*/

void    i lt_text()
{
    /*  D  one-time initializations for this module. This
     *  r utine is called from main() when the program boots.
     */

    if( !(Input_queue = makequeue( QSIZE, sizeof(CTYPE))) )
    {
            err("Not enough memory for fill buffer\n");
            exit(1);
    }
}

/*========================================================
 * Everything above this point works on char strings,
 * everything below this point works on CTYPE strings. See
 * nrmap.c for the mapping routine.
 *========================================================
 */

static  int     first_white()
{
    /* Return true if the first character in the queue
     * (the one to come out next) is a space.
     */

    CTYPE        next = *show_next(Input_queue) ;

    return( sp_used(Input_queue)  &&  WHITE(next) );
}

/*--------------------------------------------------------*/

static  int     putq(cp)
CTYPE   *cp;
{
    Last_queued = *cp & 0xff ;

    Owidth += CWIDTH( *cp );

    if( !enqueue(cp , Input_queue) )
    {
            err( "Fill buffer full\n" );
            brk();
            return 0;
    }

    return 1;
}

/*--------------------------------------------------------*/

CTYPE   *saveq()
{
    /* Save the current queue contents and flush the queue.
     * Return a pointer which, when passed to a subsequent
     * restorq() call, will restore the queue to its
     * previous state.
     */

    CTYPE       *p, *rval;

    p = (CTYPE *) malloc( (sp_used(Input_queue) +1)
                                        * sizeof(CTYPE));

    rval = p;

    if( !p )
            err("Not enough memory to save fill buffer\n");
    else
    {
            while( dequeue(p, Input_queue) )
                    ++p;

            *p      = 0;
            Owidth = 0;
    }
    return rval;
}

/*--------------------------------------------------------*/

restorq( qp )
CTYPE   *qp;
{
    /* Restore the queue to the condition it was in before
     * a previous saveq. p is a pointer returned from a
     * saveq() call. The queue is flushed before it is
     * reloaded from p.
     */

    register CTYPE      *p;

    brk();                      /*   Flush current queue     */

    if( qp )
    {
            for( p = qp ; *p ; putq( p++ ) )
                    ;

            free( qp );
    }
}

/*--------------------------------------------------------*/

prblank( n )
int     n;
{
    /* Flush the buffer and print n blank lines. This
     * routine handles the .sp command.
     *
     * If spacing is inhibited (ie .ns was given) do
     * nothing. The nospace test is done in outbuf.
     */

    while( --n >= 0 )
            outbuf( &blankline, 0 );
}

/*--------------------------------------------------------*/

add_sep( line )
CTYPE   *line;
{
    /* If there is something in the queue and that something
     * doesn't start with white space then enqueue an extra
     * space character as a word seperator.
     */

    CTYPE   c;

    if( sp_used(Input_queue) && !WHITE(Last_queued) )
    {
          c = TO_CTYPE( ' ' );
          putq( &c );
    }
}

/*--------------------------------------------------------*/

dofill( line )
CTYPE   *line ;
{
    /* Collect words until we have filled an entire line
     * and then print it.
     */

    int     pad;            /* Amount of padding needed        */
    int     tail;           /* padding needed on current line */
    int     nchars;         /* # of chars to print out         */
    int     prevwidth;      /* amount of space used by nchars */
    CTYPE   *word;          /* beginning of current word.      */
    CTYPE   *white;         /* pointer to beginning of white   */
                            /* space preceedeing current word */
    CTYPE   *p;             /* pointer into hyphenated word.   */
    CTYPE   *chop_here;
    int     extra;
    int     addhyphen;

    D( outints("dofill", line)                                   );
    D( printf(">\ndofill: prevwidth=%d, Owidth=%d\n",            )
    D(                                   prevwidth,Owidth)       );

    if( Num_center )
    {
        /* If we're doing centered lines, output the current
         * buffer without filling.
         */

        outbuf( line, 0 );
        return 0;
    }

    while( *line )
    {
        prevwidth = Owidth ;
        pad       = U_TLEN - prevwidth;
        nchars    = sp_used( Input_queue );
        white     = line ;

        /* Insert a word into the queue. Add leading blanks
         * first and then the word itself. Putq() increments
         * Owidth as necessary to reflect the amount of
         * space occupied on the output line by the
         * characters in the queue.
         */
```

(continued on page 116)

C spoken here...

High C™

Do you want to use a C compiler that

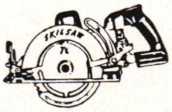
- was chosen by Ashton-Tate for dBASE III® Plus, and CAD Leader AutoDesk for AUTOCAD and AUTOSKETCH "with a twenty percent code savings over Lattice C."
- was well rated in ***Computer Language, Feb. 86:*** "Then there is High C, the most powerful compiler of all..." and ***Dr. Dobbs Journal, August 86***
- "would have saved me three weeks of porting time had I had High C instead of Microsoft's new C" *Mike LeBlanc, compiler developer, Sky Computers*
- "is the *only* C compiler for the IBM PC capable of compiling NYU's Ada/Ed compiler" *Dave Shields, research scientist, New York Univ.*
- has a complete run-time library
- has structure assignment, **enum**, **void**...
- supports nested functions as in Pascal
- supports pcc and full K&R C *plus* some latest, nifty extensions from the new ANSI-proposed C standard
- "is the highest quality C compiler for large-scale software development." *Randy Nielsen, Ansa (Paradox)*
- "saved 15% of code over five large modules of MultiMate relative to Lattice C" *David Beauchesne, Multimate International*

Pascal spoken here...

Professional Pascal™

Do you want to use a Pascal compiler that

- was chosen by Lifetree Software, Inc., for implementing Volkswriter Deluxe™
- serves as a systems and applications language at CAD/CAM giant Daisy Systems Corporation
- was well rated in ***Computer Language, May 86:*** "The clear choice for large-scale programming projects..."; and ***PC Tech Journal, July 86:*** "for team programming or for very large projects...stands absolutely alone." pg. 126 "documentation is certainly the best...a model of technical clarity." pg. 112.
- is the "howitzer" of Pascals and "could well be the most powerful Pascal compiler ever implemented on a microcomputer" ***PC Magazine, Oct. 29, 1985, p. 144***
- has 8-, 16-, and 32-bit integers; sets up to 64K bits
- has varying-strings of up to 64K characters
- has a full-fledged C macro preprocessor
- has many run-time library additions: UNIX™-like I/O, multiple heaps, interrupts, . . .
- has all the bit-pushing operators of C
- has *many* more extensions, getting you half way to Ada® for a non-Ada price



Power Tools for Power Users

Each compiler • generates **superb code**, with optimizations such as common-subexpression elimination and cross-jumping • sports no less than ***five* memory models** for the 8086 (Small, Compact, Medium, Big, and Large) • supports a unique implementation of register variables • supports the **8087/80287** in native mode, or **emulates** • supplies *three* floating-point formats • generates special instructions for the 80186/286 • generates code that runs in **80286 protected mode** • gives you **hundreds of error and warning messages**, helping you find those subtle bugs *before* you need a debugger's help • lets you **overlay *data*** as well as code (when used with PLINK86), for substantial space savings • lets you write **interrupt routines** directly in high-level language • lets you get "close to the machine" with built-in move/scan/compare operations • is supported by equivalent **resident and cross** compilers for the 80286 (UNIX V.2, Xenix, Concurrent DOS 286), 80386 (DOS, UNIX V.3), 68010/20/68881 (UNIX V & 4.2, GEM-DOS™), 32032 (UNIX 4.2), VAX (UNIX 4.2, VMS), RT PC, IBM 370,... • contains a **multi-modular cross-referencer** • produces ROM-able code for embedded applications • can talk to those **other languages** by those other vendors • gives you **64K run-time stack** space that can be shared with the heap • is endowed with an **amazing number of pragmas** (compiler controls) for customization to your application • has a compiler start-up **profile** • supports direct access to MS-DOS; library supports DOS 3.X file-sharing • generates symbolic debugger information for use with all known MS-DOS debuggers • allows **exec**-ing subprocesses • was designed for professional software developers, not hobbyists • comes with great technical support by a company that specializes in compilers • comes with extensive **typeset documentation** • and *more*... call or write for your information packet today...

Not recommended for casual use, but for applications needing **industrial-strength tools**, contact

MetaWare™
INCORPORATED
903 Pacific Avenue, Suite 201, Santa Cruz, CA 95060-4429
(408) 429-6382 (429-META), TELEX: 493-0879

Abroad:
ABC Software, The Netherlands
Microsoftware, Tokyo
Grey Matter, United Kingdom
Buchdata, Frankfurt

Professional Tools Since 1979

High C V1.3: $495 —on any MS/PC-DOS system— **Professional Pascal** V2.6: $595

OEMs: Contact us about porting our professional compilers to your systems.

TWS: Professional Compiler Developers and competitors, ask about our Translator Writing System compiler toolbox; see the review in ***Computer Language,*** *December, 1985.*

DOS Helper™: Powerful UNIX-like utilities to enhance MS-DOS, for $49.95—included free with MetaWare compilers: FIND, TAIL, MV, LS, CAT, UNIQ, FGREP, and WC.

C Validation Suite: Used by some of our competitors and many others. $2,000/Plant Site

Circle **no. 95** on reader service card.

Listing Twenty-four

(Listing continued, text begins on page 130.)

```
            if( CHAR(*line) != ' ' )     /* this is the first word */
                add_sep( line );         /* of an input line       */

            else while( CHAR(*line) == ' ' )
                putq(line++);

            chop_here = word = line;

            while(*line && CHAR(*line) != ' ')
                putq(line++);


            /* If the new word put more characters into the
             * queue than will fit on the line (U_TLEN) then
             * output everything up to (but not including)
             * the most recently added word. If hyphenation
             * is enabled, output a prefix too.
             */

            if( Owidth >= U_TLEN )
            {
                addhyphen = 0;
            if( prevwidth > (LINLEN * SPACE_SIZE) )
            {
                err("Line overflow, truncating\n");
                nchars = LINLEN;
            }
            else if( Hyphenate && *word
                            && pad > (3*SPACE_SIZE)  )
            {
                /* pad -- difference between line width
                 * and width of line up to the beginning of
                 * the previous word.
                 */

                hyphen( word, line-1 );

                p           = word;
                prevwidth += CWIDTH( TO_CTYPE('-') );

                for(; p < line &&  prevwidth < U_TLEN; p++ )
                {
                    prevwidth += CWIDTH(*p) ;

                    if( HAS_HYPHEN(*p) )
                    {
                        chop_here = p + 1;
                        addhyphen = 1;
                    }
                    else if( CHAR(*p) == '-' )
                    {
                        chop_here = p + 2;
                        addhyphen = 0 ;
                    }
                }

                nchars += (chop_here - word);
            }

            outqueue( nchars, addhyphen );
        }
    }
}

/*------------------------------------------------------------*/

brk()
{
    /* Process a line break. If there's anything in the
     * buffer, flush it out. If the adjustment mode is BOTH
     * then adjustment is turned off when the line is
     * flushed (so that the last line of a paragraph looks
     * correct. This routine assumes that the queue always
     * has less than one output lines worth of text in it.
     */

    register int        oadj;

    D( printf("Doing break\n"); )

    oadj = Adjusting;

    if( Adjmode == BOTH )
            Adjusting = 0;

    outqueue( sp_used(Input_queue), 0 );

    Adjusting = oadj ;
}

/*------------------------------------------------------------*/

outqueue( numchars, addhyphen )
int       numchars;                  /* # of chars. to dequeue */
{
    /* Output numchars characters from the input queue,
     * using outbuf(). If "addhyphen" is true then the a
     * hyphen is printed at the end of the line.
     */

    register CTYPE       *p ;
    static   CTYPE       buf[MAXSTR] ;
    CTYPE                ochar ;

    D(printf("outqueue: putting %d characters\n",numchars));

    if( numchars <= 0 )
            return;

    p = buf;

    while( --numchars >= 0  && dequeue(&ochar,Input_queue))
    {
            Owidth -= CWIDTH( ochar );
            *p++ = ochar;
    }

    --p;

    /* Delete trailing white space from the output buffer
     * and leading white space from the queue.
     */

    while( WHITE(*p) )
            if( --p < buf )
                    break;
    *++p = '\0';

    while( first_white() && dequeue(&ochar, Input_queue) )

            --Owidth ;

    outbuf( buf, addhyphen );           /* Output the line */
}
```

End Listing Twenty-four

Listing Twenty-five

```
#         Makefile for Lattice lmk to  manufacture nr using
#         the Microsoft C compiler, version 4.0
#
CV_CSWITCH = /Zi
CV_LINKSW  = /CO /M

CSWITCH    =
LINKSW     =  /NOI /STACK:4096

OBJ1 = nr.obj nrcmd.obj nreg.obj nrexcept.obj nrglbls.obj
OBJ2 = nrhyphen.obj nrinp.obj nrmac.obj nrmap.obj nrmsc.obj
OBJ3 = nrout.obj nrprocs.obj nrtab.obj nrtext.obj

# ---------------------------------------------------------

.c.obj:
        cc -c $(CV_CSWITCH) $(CSWITCH) $*.c >>err

# ---------------------------------------------------------

nr.exe:         $(OBJ1) $(OBJ2) $(OBJ3)
                link $(CV_LINKSW) $(LINKSW) <@<
$(OBJ1) +
$(OBJ2) +
$(OBJ3)
nr
nr
\lib\tools.lib
<

final:  $(OBJ1) $(OBJ2) $(OBJ3)
                link $(LINKSW) <@<
$(OBJ1) +
$(OBJ2) +
$(OBJ3)
nr
nr
\lib\tools.lib
<

# ---------------------------------------------------------

nr.obj:         nr.c          nr.h
nrcmd.obj:      nrcmd.c       nr.h
nreg.obj:       nreg.c        nr.h
nrglbls.obj:    nrglbls.c     nr.h
nrhyphen.obj:   nrhyphen.c    nr.h     nrmap.h
nrinp.obj:      nrinp.c       nr.h
nrmac.obj:      nrmac.c       nr.h
nrmap.obj:      nrmap.c       nr.h     nrmap.h
nrmsc.obj:      nrmsc.c       nr.h
nrout.obj:      nrout.c       nr.h     nrmap.h    nrtlen.h
nrprocs.obj:    nrprocs.c     nr.h
nrtab.obj:      nrtab.c       nr.h
nrtext.obj:     nrtext.c      nr.h     nrmap.h    nrtlen.h
nrexcept.obj:   nrexcept.c    nr.h     nrmap.h
```

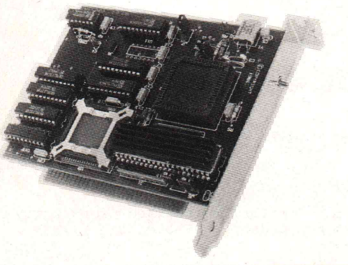
End Listings

(Listing 26 will be in next issue.)

MICROWAY MEANS 8087 PERFORMANCE

FastCACHE-286™

Runs the 80286 at 8.5 or 11 MHz and the 80287 at 5, 6 or 11 MHz. Includes 8 kbytes of 55ns CACHE. Works with more PCs than any other accelerator, including Leading Edge Model D, Compaq, and Turbo motherboards. Includes 8088 Reboot Switch, DCache and Diagnostics........................ **From $449**



LOTUS/INTEL EMS SPECIFICATION BOARDS

MegaPage™ The only EMS board which comes populated with two megabytes of cool-running, low power drain CMOS RAM installed. Includes RAM disk, print spooler, disk cache and EMS drivers. For the IBM PC, XT and compatibles...**$549**

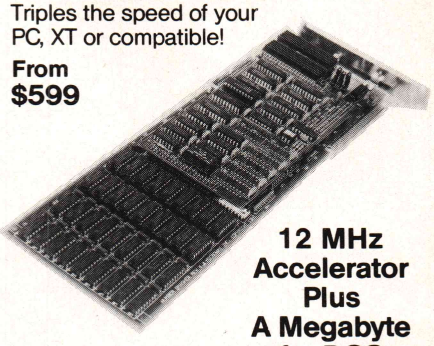
MegaPage with ØK.............. **$149**

MegaPage AT/ECC™ EMS card for the PC AT and compatibles includes Error Correction Circuitry. With ECC, 11 RAM chips cover 256K so the user never encounters RAM errors. Sold populated with 1 megabyte CMOS . . . **$699** or with 3 megabytes CMOS cool running low power drain RAM . . . **$1295**. Optional serial/parallel daughterboard....... **$95**

NUMBER SMASHER/ECM™

Triples the speed of your PC, XT or compatible!

From $599



12 MHz Accelerator Plus A Megabyte for DOS

***PC Magazine* "Editor's Choice"**

DATA ACQUISITION and REAL TIME TOOLS

DAL™ – "Data Acquisition Language."

Unkelscope™ – A real time data acquisition, control and process software pkg.

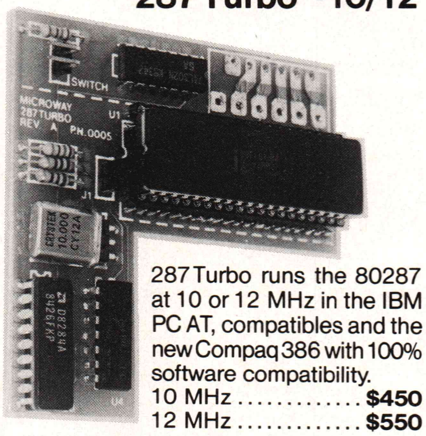
87 FFT and 87 FFT-2

TransView Menu driven FFT Spectrum/transfer analyzer.................... **$250**

RTOS – REAL TIME OPERATING SYSTEM A multi-user, multi-tasking real time operating system. Includes a configured version of Intel's iRMX-86, LINK-86, LOC-86, LIB-86, OH-86 and the MicroWay 87DEBUG. Runs on the IBM-PC, XT, PC-AT and COMPAQ.................. **$600**

INTEL COMPILERS Available for RTOS FORTRAN-86, PASCAL-86, PL/M-86.

287 Turbo™-10/12



287 Turbo runs the 80287 at 10 or 12 MHz in the IBM PC AT, compatibles and the new Compaq 386 with 100% software compatibility.
10 MHz............. **$450**
12 MHz............. **$550**

***PC Magazine* "Editor's Choice"**

8087 UPGRADES

All MicroWay 8087s include a one year warranty, complete MicroWay Test Program and installation instructions.

8087 5 MHz.................. **$114**
For the IBM PC, XT and compatibles

8087-2 8 MHz............... **$149**
For Wang, AT&T, DeskPro, NEC, Leading Edge

80287-3 5 MHz............. **$179**
For the IBM PC AT and 286 compatibles

80287-6 6 MHz............. **$229**
For 8 MHz AT compatibles

80287-8 8 MHz............. **$259**
For the 8 MHz 80286 accelerator cards

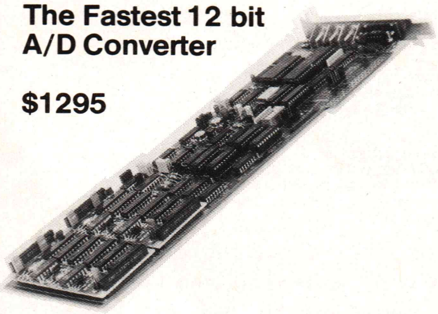
80287-10 10 MHz.......... **$395**
For the Compaq 386

Call for prices on V20, V30, 64K, 128K and 256K RAM

A2D-160™

The Fastest 12 bit A/D Converter

$1295



160,000 Samples per second
Pseudo Random Noise Generator/DAC
Optional signal conditioners
AFM-50 Programmable Low Pass Filter Module..................... **$225**

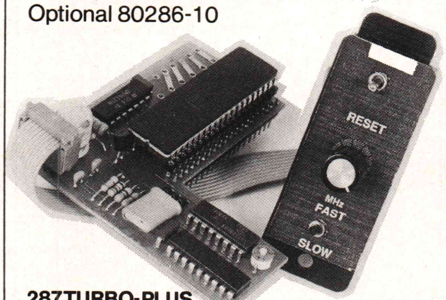
8087 SOFTWARE

IBM BASIC COMPILER........... **$465**
MICROSOFT QUICK BASIC....... **$79**
87BASIC COMPILER PATCH..... **$150**
IBM MACRO ASSEMBLER....... **$155**
MS MACRO ASSEMBLER......... **$99**
87MACRO/DEBUG............... **$200**
MICROSOFT FORTRAN.......... **$209**
RM FORTRAN.................... **$399**
LAHEY FORTRAN F77L......... **$477**
MS or LATTICE C................ **CALL**
STSC APL★PLUS/PC............ **$450**
STSC STATGRAPHICS........... **$675**
SPSS/PC+....................... **$675**
87SFL Scientific Functions....... **$250**
PHOENIX PRODUCTS........... **CALL**
FASTBREAK for 1-2-3 V.1A........ **$79**
HOTLINK for 1-2-3 V.1A........... **$99**

287 TURBO-PLUS™

Speeds up your AT

Adjustable 80286 Clock 6-12 MHz
10 MHz 80287 Clock
Plus Full Hardware Reset......... **$149**
Optional 80286-10



287TURBO-PLUS
With 80287 10 MHz.................... **$549**
With 80287 12 MHz.................... **$629**

CALL (617) 746-7341 FOR OUR COMPLETE CATALOG

MicroWay
P.O. Box 79
Kingston, Mass.
02364 USA
(617) 746-7341

The World Leader in 8087 Support!

Circle **no. 300** on reader service card.

MicroWay Europe
32 High Street
Kingston-Upon-Thames
Surrey England KT1 1HL
Telephone: 01-541-5466

Listing One *(Text begins on page 140.)*

```
( Elemental tools                                   Ham 10:31 12/13/86 )

: -CUR  14  0 SET-CURSOR ;  ( no cursor )
: +CUR   6  7 SET-CURSOR ;  ( normal cursor )

: BACK   ( n - ) 0 ?DO 8 EMIT LOOP ;  ( backspace word )

  0 CONSTANT NEW   ( to collect new digits )
 -1 CONSTANT OLD   ( to provide existing number to routine )

: INCR  ( a - )   1 SWAP +! ;  ( increments variable )
: DECR  ( a - )  -1 SWAP +! ;  ( decrements variable )

: Bs? ( n - f )  8 = ;     ( T if backspace pressed )
: Cr? ( n - f ) 13 = ;     ( T if carriage return pressed )

  VARIABLE OK-NEG   ( T allows for entry of - ; F rejects - )
  VARIABLE SOUND    ( T if using sound )

: BELL ( - ) SOUND @  IF   440  8 BEEP ( short beep ) THEN ;


  CREATE #PAD 15 ALLOT  ( work area )

: #P! ( c n - )  #PAD + C! ; ( stores character c at offset n )

  CREATE #VAR 14 ALLOT      ( holds various values              )

  #VAR        CONSTANT #DEC   ( no. of fractional digits ALLOWED )
  #VAR  2+    CONSTANT #dec   ( no. of fractional digits ENTERED )
  #VAR  4 + CONSTANT #WHOLE ( no. of whole digits entered       )
  #VAR  6 + CONSTANT #HIT   ( no. of keystrokes                 )
  #VAR  8 + CONSTANT NEG~   ( T if number is negative           )
  #VAR 10 + CONSTANT dec~   ( T if decimal point entered        )
  #VAR 12 + CONSTANT DIGCNT ( counts no. of digits for old nos.)

: PLACES ( n - ) #DEC ! ; ( sets # of decimal places allowed )

: #init  #dec 12 ERASE ( don't erase #DEC ) #PAD 15 ERASE ;

: *HIT/NEG  #HIT 4 ERASE ;  ( resets no. hit and negative flag )

: NEG? ( - f )  NEG~ @ ; ( T if number is negative )

: dec? ( - f )  dec~ @ ; ( T if dec point entered )


( Get and edit keystroke                            Ham 10:33 12/13/86 )

: CAPITALIZE ( c - C )  DUP 96 > OVER 123 < AND IF BL - THEN ;

: FIXUP ( c-c) DUP ASCII B = OVER BL = OR IF DROP ASCII C THEN
        ( convert B and space bar to C := clear number entry )
        ( L := 1 ) DUP ASCII L = IF DROP ASCII 1 THEN
        ( O := 0 ) DUP ASCII O = IF DROP ASCII 0 THEN ;

: #?  ( n - f ) DUP ASCII / > SWAP ASCII : < AND ; ( T if digit)

: BAD? ( n - f ) DUP #?  OK-NEG @  IF OVER ASCII - = OR THEN
                         #DEC   @  IF OVER ASCII . = OR THEN
           OVER ASCII C = OR  OVER Bs? OR  SWAP Cr? OR  NOT ;

: GET#  ( - n) BEGIN KEY CAPITALIZE FIXUP DUP BAD?
               WHILE DROP BELL REPEAT ;


( Collection box                                    Ham 10:34 12/13/86 )

: #,S ( #w - #, ) 3 /MOD SWAP 0= + 0 MAX ;
  ( takes # of whole-number digits, leaves # of commas required)
  ( Warning:  Assumes 83-Std flag = -1; negate flag if 79 Std )

: FULLCNT ( n - n' ) #DEC @ IF 1+ THEN  OK-NEG @ IF 1+ THEN ;
    ( adds to char cnt the decimal point and minus sign if any )

: BOXSIZE ( n - m ) DUP ( # of digits ) #DEC @ - ( #whole digts)
   DUP 1 < ( T if no whole digits ) NEGATE ( 83-Std flag ) >R
   #,S ( # of commas )  R> + +   2+ ( space at either end )
   FULLCNT ; ( leaves number of character in box )

: BOX   ( n - ) BOXSIZE SPACES ;
       ( prints inverse spaces to define field for number entry)


( Sign/decimal                                      Ham 10:34 12/13/86 )

: -.  ( displays - or . or both when no digits yet entered )
      NEG? dec? AND IF 3 BACK ." -. "
                    ELSE NEG? IF 2 BACK ." - "
                              ELSE dec? IF 2 BACK ." . "
      THEN THEN THEN ;


( Count digits; show number                         Ham 10:35 12/13/86 )

: 2, ( d - ) , , ;  ( store double into dictionary )

  CREATE NINES  9. 2,  99. 2,  999. 2,  9999. 2,  99999. 2,
   999999. 2,  9999999. 2,  99999999. 2,  999999999. 2,

: #OFDIGITS ( d - # ) DABS 1 DIGCNT !
      BEGIN 2DUP DIGCNT @ 1- 4 * NINES + 2@ D>
      WHILE DIGCNT INCR REPEAT 2DROP DIGCNT @ ;

: PUT#  ( - adr cnt ) ( prepares number for display )
   0. #PAD 1- CONVERT DROP 2DUP #OFDIGITS >R
   <#  dec?  IF #dec @   0 ?DO # LOOP ASCII . HOLD THEN
       R> #dec @ - #,S  0 ?DO # # #  ASCII , HOLD LOOP
       #S NEG? SIGN #> ;


( Display the number nicely                         Ham 19:39 12/04/86 )

: DISPLAY# ( n - n ) DUP ( get another copy of max # of digits )
   BOXSIZE DUP BACK
   ( back up to beginning of entry field; top of stack is )
   ( size of box, which is greater than # of digits )
   #HIT @
    IF 1- ( space at end ) PUT# ROT OVER - SPACES TYPE SPACE
    ELSE SPACES ( new box ) -. THEN  ;
   ( n is max no. of digits to be entered, which stays on stk )


( Wrap-up routine                                   Ham 21:15 11/27/86 )

: 10D*  ( d - 10*d ) 2DUP 2DUP D+ 2DUP D+ D+ 2DUP D+ ;

: SCALE#  #DEC @ ?DUP IF #dec @ -  0 ?DO 10D* LOOP THEN ;
  ( scale up to integer from decimal fraction )

: #DONE  ( - d # )
    ( leaves double number entered and no. of digits entered )
    ( no. of digits = zero means no digits entered )
    0. #PAD 1- CONVERT ( leaves addr of 1st nonconverting char )
    #PAD - ( number of digits ) >R
    NEG? IF DNEGATE THEN   SCALE#  R> DUP 0=
       IF ( number is 0, see whether key pressed or no entry )
         DROP #HIT @ 0> NEGATE ( Note:  83-Std flag ) THEN ;


( Adjust counts                                     Ham 18:51 11/06/86 )

: #dec-ADJ    #dec @ IF #dec DECR THEN ; ( down one decimal )

: #WHOLE-ADJ  dec? NOT IF #WHOLE DECR THEN ; ( down 1 whole no.)

: NO-.? ( - f ) #HIT @ 0= NEG? 0= dec? 0= AND AND ;


( When decimal point is hit                         Ham 18:53 11/06/86 )

: .ROUTINE  dec? IF   BELL     ( decimal point already entered )
                 ELSE dec~ ON ( mark entry of decimal point )
                 THEN ;


( Check if need to adjust digits                    Ham 18:23 11/06/86 )

 ( WHOLE-CK & DEC-CK have this stack diagram: ( n f - n f' )
 ( where n is the no. of digits entered so far )

: WHOLE-CK dec? 0= IF OVER #DEC @ - #WHOLE @ = OR THEN ;
  ( makes flag T if dec pt not entered AND we have all the )
  ( whole number digits that we can accept )

: DEC-CK   #DEC @ ?DUP IF #dec @ = OR THEN ;
  ( makes flag T if we have all the digits to the right )
  ( of the decimal that we can accept )

  ( The true flag will cause the latest digit entered to be )
  ( dropped and the bell to sound (if SOUND is on)


( Count each digit entered                          Ham 18:24 11/06/86 )

  VARIABLE 0START ( T if starting with whole number zero )

  ( A starting whole number value of zero is in effect a      )
  ( leading zero and should not be counted in the total of    )
  ( digits entered, or else the final numeric digit will not )
  ( be accepted. )

: CNT-DIGIT dec? IF   #dec INCR
                 ELSE 0START @ IF   0START OFF ( 1-time switch )
                                ELSE #WHOLE INCR THEN THEN ;


( Initialization for "old" numbers                  Ham 18:26 11/06/86 )

  ( If old number is decimal, all places are present. )

: SET-dec  #DEC @ ?DUP IF #dec ! dec~ ON THEN ;

: SET-NEG ( d n - n d ) ROT ROT ( move dbl to top ) 2DUP 0. D<
     IF ( neg: convert and note sign ) DNEGATE NEG~ ON THEN ;

  ( Put number into #PAD as an string of ASCII values: )

: SET-#P ( d - ) <# dec? IF #dec @ 0 DO # LOOP THEN
    DIGCNT @ #DEC @ > IF #S THEN #> #PAD SWAP CMOVE ;


( Initializes for loop                              Ham 18:34 11/06/86 )

: DSET ( d # T|# F -- m n p )
  ( m = # of digits to collect, n p = limits for loop )
```

```
  0START OFF #init OVER BOX
  IF ( old number present ) SET-dec SET-NEG 2DUP
    2DUP OR 0= #DEC @ 0= AND ( double is both zero and whole )
      IF 0START ON THEN       ( so mark it as a zero start     )
    #OFDIGITS #DEC @ MAX DUP #HIT ! ( set # of digits entered )
    DUP #DEC @ - 0 MAX #WHOLE !      ( set # of whole digits ent)
    ROT ROT SET-#P                   ( make & save ASCII string )
    SWAP DUP 1+ ROT 0START @ +       ( using 83-Std flag to decr)
  ELSE ( no old number present ) DUP 1+ 0 THEN  ;


( Backspace routine                               Ham 10:35 12/13/86 )

  VARIABLE INDX ( holds index from loop )

: "I" ( - index ) INDX @ ;  ( lets me get I from outside loop )

: NO#?  ( - f) #HIT @ 0= ;  ( T = no digits entered )

: BSP-ROU ( -- loop-incr ) dec? #dec @ 0= AND
    IF dec~ OFF 0  ( just backed over the decimal point )
    ELSE "I" IF 0 "I" 1- #P! ( zap previous entry in string )
                #HIT DECR #dec-ADJ #WHOLE-ADJ ( adjust counts )
                NO-.?   ( no minus sign or decimal point? )
                    IF   BELL "I" NEGATE  ( back up all the way )
                    ELSE NO#? IF   "I" NEGATE
                              ELSE -1 THEN THEN
            ELSE *HIT/NEG BELL 0 THEN THEN ;

( The above above takes care of the details of the backspace in
  numeric entry & leaves the proper loop increment on the stk )


( Final input word                                Ham 18:51 11/06/86 )

: DIGITS ( d # T | # F -- d #) REVERSE DSET DO DISPLAY# GET#
      DUP Bs?          IF DROP I INDX !  BSP-ROU
 ELSE DUP ASCII - =  IF DROP NEG~ @ NOT NEG~ ! 0
 ELSE DUP ASCII . =  IF DROP .ROUTINE 0
 ELSE DUP ASCII C =  IF DROP #PAD C@ ASCII 0 <>
                        IF #HIT @ NEGATE  ELSE 0 THEN  #init
 ELSE DUP I #P! ( store char )  Cr? IF LEAVE THEN
      I 1+ #HIT ! ( count of net keystrokes )
      DUP ( # of digits to enter ) I = WHOLE-CK DEC-CK
          IF ( at end: reject digit )  0 I #P!  I #HIT ! BELL 0
          ELSE #PAD I + C@ ASCII 0 <> I 0<> OR dec? OR
               NEGATE ( 83 Std flag )  DUP  IF CNT-DIGIT THEN
```

```
  THEN THEN THEN THEN THEN +LOOP
DROP ( count ) #DONE REVERSE ;


( Test                                          Ham 18:51 11/06/86 )

  0 PLACES
  OK-NEG OFF
  SOUND ON
  -CUR

  5 NEW DIGITS

  CR CR
  2 PLACES
  OK-NEG ON

  7 NEW DIGITS

  +CUR
```

End Listing

Subscribe to Micro/Systems Journal

Yes! I want to subscribe to
Micro/Systems Journal
and save over 15% off the cover price.

__ **1 Year (6 issues) $20** __ **2 Years (12 issues) $35**

__Please charge my: __ Visa __ MC __ American Express
__ Payment enclosed __ Bill me later

Card # __________ Exp. date __________
Signature __________
Name __________
Address __________
City __________ State _____ Zip __________

Canada & Mexico add $3 for surface mail; $7 for airmail. All other countries add $12 for airmail. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.
A Publication of M & T Publishing, Inc.

300

Yes! I want to subscribe to
Micro/Systems Journal
and save over 15% off the cover price.

__ **1 Year (6 issues) $20** __ **2 Years (12 issues) $35**

__Please charge my: __ Visa __ MC __ American Express
__ Payment enclosed __ Bill me later

Card # __________ Exp. date __________
Signature __________
Name __________
Address __________
City __________ State _____ Zip __________

Canada & Mexico add $3 for surface mail; $7 for airmail. All other countries add $12 for airmail. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.
A Publication of M & T Publishing, Inc.

30

Yes! I want to subscribe to
Micro/Systems Journal
and save over 15% off the cover price.

__ **1 Year (6 issues) $20** __ **2 Years (12 issues) $35**

__Please charge my: __ Visa __ MC __ American Express
__ Payment enclosed __ Bill me later

Card # __________ Exp. date __________
Signature __________
Name __________
Address __________
City __________ State _____ Zip __________

Canada & Mexico add $3 for surface mail; $7 for airmail. All other countries add $12 for airmail. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.
A Publication of M & T Publishing, Inc.

3

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

For the Advanced Computer User
Micro/Systems Journal™
501 GALVESTON DR.
REDWOOD CITY, CA 94063

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

For the Advanced Computer User
Micro/Systems Journal™
501 GALVESTON DR.
REDWOOD CITY, CA 94063

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

For the Advanced Computer User
Micro/Systems Journal™
501 GALVESTON DR.
REDWOOD CITY, CA 94063

Subscribe to Micro/Systems Journal

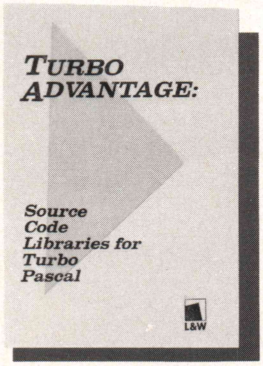
M&T Books and Software Tools

inside:

- Turbo Pascal Tools
- Toolbook of Forth
- C Toolbox
- 68000 Programming
- Z80 Programming
- Nr: An Nroff-like Text Formatter
- Interfacing to MS-DOS
- DDJ Back Issues
- DDJ Listings on Disk
- Dr. Dobb's Bound Volumes

M & T BOOKS AND SOFTWARE TOOLS

Programming Eloquence



TURBO Advantage:

Source Code Library for Turbo Pascal

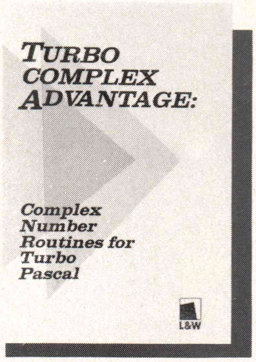
This library of more than 220 routines—complete with source code, sample programs and documentation—will save you hours of work developing and optimizing your programs!

Routines are organized and documented under the following categories: bit manipulation, file management, MS-DOS support, sorting, string operations, arithmetic calculations, data compression, differential equations, Fourier analysis and synthesis, matrices and vectors, statistics, and much more! All source code is included.

A detailed manual includes a description of the routine, an explanation of the methods used, the calling sequence, and a simple example. For MS/PC-DOS systems.

TURBO Advantage: Source Code Libraries for Turbo Pascal **is also available with** ***TURBO Advantage Complex: Complex Number Routines for Turbo Pascal*** **and** ***TURBO Advantage Display: Form Generator for Turbo Pascal.***

Turbo Advantage Item #070 $49.95



TURBO Advantage Complex:

Complex Number Routines for Turbo Pascal

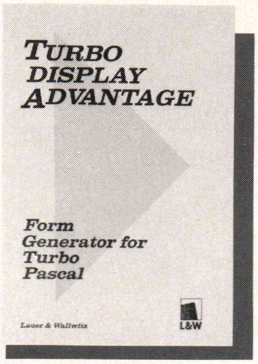
Working with complex numbers is easy with the Turbo Pascal procedures and routines provided in TURBO Advantage Complex!

TURBO Complex provides procedures for performing all the arithmetic operations and necessary real functions with complex numbers. Each procedure is based on predefined constants and types. By using these declarations the size of arrays are easily adapted. Each type declaration is a record with both a real and imaginary part. Use these procedures to build more sophisticated functions in your own programs.

TURBO Complex also demonstrates the usage of these procedures in routines for vector and matrix calculation with complex numbers and variables; simultaneous Fourier transforms; calculations of convolution and correlation functions; low-pass, high-pass, band- pass and band-rejection digital filters; and in solving linear boundary-value problems.

Source code and documentation is included for MS-DOS systems. Some of the *TURBO Complex* routines are most effectively used with routines contained in *TURBO Advantage.*

TURBO Advantage/ Complex Package	**Item #070A**	**$115**
TURBO Complex	**Item #071**	**$89.95**



TURBO Advantage Display:

Form Generator for Turbo Pascal

Now, even if you have little programming knowledge, you can design and process forms to fit your needs!

TURBO Display includes a menu-driven form processor, 30 Turbo Pascal procedures and functions to facilitate linking created forms to your program, and full source code and documentation. For MS-DOS systems.

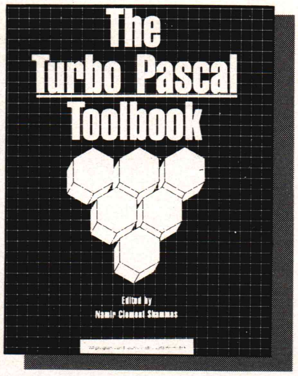
Some of the *TURBO Advantage: Source Code Libraries for Turbo Pascal* routines are necessary to compile *TURBO Display.* You save $20 when you order *TURBO Advantage: Source Code Libraries for Turbo Pascal* together with *TURBO Display: Form Generator for Turbo Pascal.* Receive both for only $99.95!

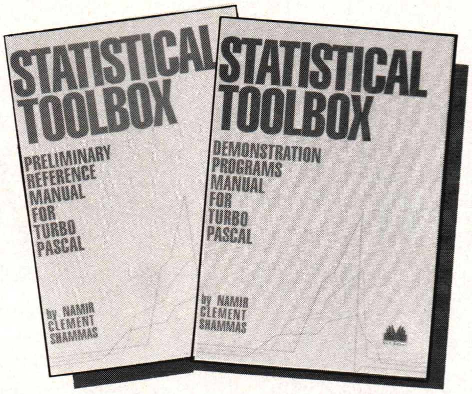
TURBO Display: Form Generator for Turbo Pascal is also available individually for $69.95.

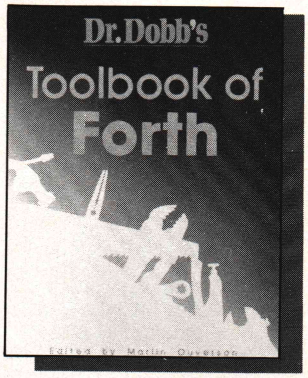
TURBO Advantage/ Display Package	**Item #070B**	**$99.95**
TURBO Display	**Item #072**	**$69.95**

M & T BOOKS AND SOFTWARE TOOLS

Programming Eloquence







The Turbo Pascal Toolbook

Edited by Namir Clement Shammas

Make your programming easier and more powerful with *The Turbo Pascal Toolbook!*

You'll find:

- an extensive library of low-level routines
- external sorting and searching tools, presenting a new database routine that combines the best features of the B-tree, B+ and B++ trees
- window management, to help you create, sort, and overlay windows
- artificial intelligence techniques
- mathematical expression parsers, offering two routines that convert mathematical expressions into RPN tokens
- a smart statistical regression model that searches for the best regression model to represent a given set of data.

All routine libraries and sample programs are on disk for MS-DOS systems, and over 800K of Turbo Pascal source code is included!

Turbo Pascal Toolbook
Item #080 **$25.95**
Turbo Pascal Toolbook with disk
Item #081 **$45.95**

STAT Toolbox

for Turbo Pascal

Bring convenience, power, and versatility to your statistics programs!

Two statistical packages in one!

A library disk and reference manual
Use these powerful statistical routines to build your applications.
Routines include:

- statistical distribution functions
- random-number generation
- basic descriptive statistics
- parametric and non-parametric statistical testing
- bivariate linear regression, multiple and polynomial regression.

A demonstration disk and manual
This package incorporates the library of routines into a fully functioning statistical program. Two data management programs are included to facilitate the storage and maintenance of data.

Full source code is included. (For IBM PC's and compatibles. Turbo Pascal version 2.0 or later, and PC DOS 2.0 or later are required).

STAT Toolbox **Item #050** **$69.95**

Dr. Dobb's Toolbook of Forth

This comprehensive collection of useful Forth programs and tutorials contains *DDJ's* best Forth articles, expanded and revised along with new material. You'll find sections on:

- Mathematics in Forth
- Modifications/Extensions
- Forth Programs
- Forth—the Language
- Implementing Forth

You'll also find Appendixes that will help you convert fig Forth to Forth-83, and tell you how to stay up-to-date on the latest developments and refinements of this popular language.

The screens in the book are also available on disk as ASCII files.

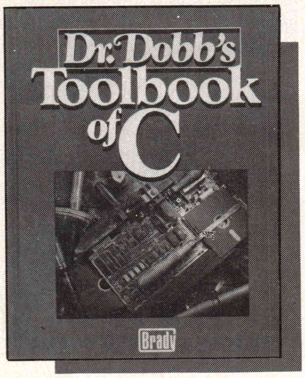
Dr. Dobb's Toolbook of Forth
Item #030 **$22.95**
Dr. Dobb's Toolbook of Forth with Disk
Item #031 **$39.95**

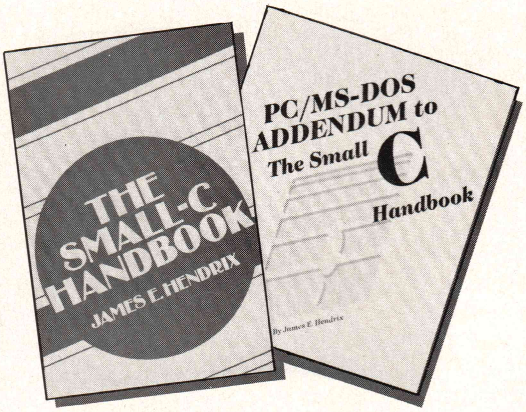
Please specify MS/PC-DOS, Apple II, Macintosh, or CP/M. For CP/M disks, specify Osborne or 8" SS/SD.

CALL
800-533-4372
TOLL FREE

In CA 800-356-2002

C Toolbox





Dr. Dobb's Toolbook of C

This authoritative reference contains over 700 pages of the best C articles and source code from *Dr. Dobb's Journal,* along with new material by C experts. The level is sophisticated and pragmatic, appropriate for professional C programmers. You'll find hundreds of pages of valuable C source code, including a complete compiler, an assembler, and text processing utilities, all available on disk.

Toolbook of C Item #005 $29.95

Small-C Compiler

Like a home-study course in compiler design, *the Small-C Compiler* and the *Small-C Handbook* provide everything you need for learning how compilers are constructed, and for learning C at its most fundamental level. Full source code is included.

CP/M Small-C Compiler with Handbook Item #006B $37.90

MS/PC-DOS Small-C Compiler with Handbook and Addendum Item #006C $42.90

Small-C Compiler Item #007 $19.95

Small-Tools: Programs for Text Processing

These Small-C programs perform specific, modular operations on text files, including: editing, formatting, sorting, merging, listing, printing, searching, changing, transliterating, copying, concatenating, encrypting and decrypting, and more. Supplied as source code. With the *Small-C Compiler* you can select and adapt these tools to meet your own needs. Documentation is included.

Small-Tools Item #010A $29.95

Small-Mac: An Assembler for Small-C

This assembler includes: a simplified macro facility, C language expression operators, object file visibility, descriptive error messages, and an externally defined instruction table. You'll receive the macro assembler, linkage editor, load-and-go loader, library manager, CPU configuration utility, and a utility to dump relocatable files. Documentation is included. For CP/M systems only.

Small-Mac Item #012A $29.95

The Small-C Handbook

by James E. Hendrix

James E. Hendrix's *Small-C Handbook* is the only complete reference for the Small-C language and Small-C Compiler. In addition to describing the operation of the compiler, the book discusses assembly language concepts and program translation tools, and even how to generate a new version of the Small-C Compiler, available on disk. *The Small-C Handbook* is also available with an addendum for the MS/PC-DOS version.

Small-C Handbook Item #006 $19.95

Small-C Handbook with MS/PC-DOS Addendum Item #006A $22.95

Small-Windows: A Windowing Library for Small C

Small-Windows is a complete windowing library for Small-C. The package contains: 18 video functions written in assembly language; 7 menu functions that support both static and pop-up menus; and 41 window functions, including functions to clean, frame, move, hide, show, scroll, push, and pop windows. A file directory facility illustrates the use of the window and menu functions and provides file selection, renaming and deletion capability. Two test programs are also included. For PC/MS-DOS systems only. Documentation and full source code is included.

Small-Windows Item #109 $29.95

Please indicate MS/PC-DOS or CP/M. For CP/M, specify: Apple, Osborne, Kaypro, Zenith Z-100 DS/DD, 8″ SS/SD.

M & T BOOKS AND SOFTWARE TOOLS

CP/M C Package

Save over $27!

Receive: *Dr. Dobb's Toolbook of C, The Small-C Handbook, Small-C Compiler, Small-Mac Assembler,* and *Small-Tools Text Processing Programs.* Only $99.95!

CP/M C Package   Item #005A $99.95
Please specify: Apply, Osborne, Kaypro, Zenith Z-100 DS/DD, or 8″ SS/SD

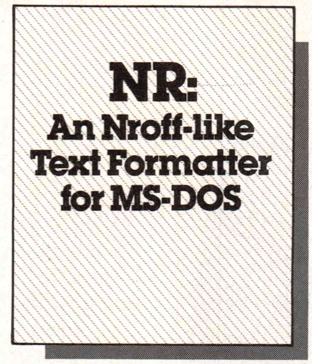
MS/PC-DOS C Package

Save $22!

Receive: *Dr. Dobbs Toolbook of C, The Small-C Handbook* and *MS/PC-DOS Addendum, Small-C Compiler, Small-Tools Text Processing Programs* and *Small Windows.* Only $109.95!

MS/PC-DOS C Package
Item #005W $109.95

Programming Tools



Nr: An Nroff-like Text Formatter for MS-DOS

Nr is an expanded version of the text formatter described in *Dr. Dobb's* February through April 1987 issues. Nr is written in C and is compatible with the Unix Nroff. You'll find complete implementation of the -ms macro package, and an in-depth description of how -ms works.

Nr does hyphenation and simple proportional spacing. It supports automatic Table of Contents and Index generation, automatic footnotes and endnotes, italics, boldface, overstriking, understriking, and left and right margin adjustment. Nr also contains:

- extensive macro & string capability
- number registers in various formats including roman numerals and arabic, spelled out and in outline form
- diversions and diversion traps
- input and output line traps

Nr comes configured for any Diablo-compatible printer, and Hewlett Packard's ThinkJet and LaserJet. It is easily configurable for most other printers and comes with full source code so that you can make it work with your system.

For PC compatibles.

Nr   Item #165 $29.95

Interfacing to MS-DOS

by William Wong

Originally featured in *Micro/Systems Journal,* **Interfacing to MS- DOS** provides ten concise articles that will orient any experienced programmer to the MS-DOS environment. **All source code discussed is also contained on disk.**

Topics include: program construction, character base input and output functions, and file access. You'll also find a discussion of CP/M style vs. Unix-style DOS file access, sample program files, and a detailed description of how to build device drivers. A device driver for a memory disk and a character device driver are provided on disk with full source code.

Interfacing to MS-DOS   Item #166 $29.95

In CA 800-356-2002

DDJ 1986/87 Back Issues

March 1986 #113 Volume XI, Issue 3
Parallel Processing—Concurrency and Turbo Pascal—What Makes DOS Fast—Minimizing Arbitrary Functions—MC68000 vs. NS32000.

April 1986 #114 Volume XI, Issue 4
Special AI Issue—Programming in LISP and Prolog—An Expert at Life—Perils of Protected Mode—I/O Redirection for the Shell.

May 1986 #115 Volume XI, Issue 5
Software Design form the Outside In—Dan Bricklin's DEMO Program—Cryptographer's Toolbox—EGA Graphics & Fact PC Graphics—How to Write Memory Resident Code.

June 1986 #116 Volume XI, Issue 6
Telecommunications Without Errors—General-Purpose Sorting—Structured Programming.

Aug. 1986 #118 Volume XI, Issue 8
Special C Issue—Benchmarking C Compilers—The Joy of Conciseness—Nearly Perfect Trees—Generics in Ada—Real-World Data Types.

Sept. 1986 #119 Volume XI, Issue 9
Smooth Algorithms—MS-DOS Directory Traversal—Turbo Boards Review—Radix Sort—Does Turbo Prolog Measure Up—Crawling Memory Test.

Oct. 1986 #120 Volume XI, Issue 10
80386 Programming—MS-DOS File Browsing—Converting to the 320xx—Modula-2 Compiler Review—Factoring in Forth.

Nov. 1986 #121 Volume XI, Issue 11
Graphics Routines—The New Graphics Chips—Programming Tips in C, Modula-2, Pascal, and Ada—68K Graphics.

Dec. 1986 #122 Volume XI, Issue 12
Multitasking—32000 Assembler—Comparing String Comparisons—Turbo Pascal Procedural Parameters.

Jan. 1987 #123 Volume XII, Issue 1
Annual 68K Issue—68K Mini Forth OS-9 Operating System—Mac and Amiga Interface Programming.

Feb. 1987 #124 Volume XII, Issue 2
Editors and Assemblers.

Mar. 1987 #125 Volume XII, Issue 3
Data Compression Techniques.

Other issues are also available. Please inquire.

Dr. Dobb's 1987 Listings

You may also receive on disk all the source code for articles in the following issues. Available formats: MS-DOS, Macintosh, Kaypro.

January 1987	**Item #123**	**$14.97**
February 1987	**Item #124**	**$14.97**
March 1987	**Item #125**	**$14.97**
April 1987	**Item #126**	**$14.97**

Dr. Dobb's 1986 Listings #1

JANUARY—APRIL 1986
Includes listings from the following articles, and more.
January Issue #111
"A Simple OS for Realtime Applications; 68000 Assembly Language Techniques for an Operating System Kernel" by *DDJ* editor Nick Turner.
February Issue #112
"Data Abstraction with Modula-2" by Bill Walker and Stephen Alexander
March Issue #113
"Concurrency and Turbo Pascal: An Approach to Implementing Coroutines in Pascal" by Ernest Bergmunn.
April Issue #114
"Boca Raton Inference Engine; Lsp, Prolog, and Expert 2 Techniques and Code" by Robert Brown.

Dr. Dobb's Listings #1/86 **Item #170 $14.95**

Dr. Dobb's 1986 Listings #2

MAY—AUGUST 1986
Includes listings from the following articles, and more.
May Issue #115
"Simple Plots with the Enhanced Graphics Adapter" by Nabajyoti Barkukati.
June Issue #116
"Compuserve B Protocol" by Steve Wilhite.
July Issue #117
"Structured Programming; Tiny Tools, Array-Defining Words" by Michael Ham.
August Issue #118
"Structured Programming; Generic Routines in Ada and Modula-2. Extended For Loop" by Namir Shammas.

Dr. Dobb's Listings #2/86 **Item#171 $14.95**

Dr. Dobb's 1986 Listings #3

SEPTEMBER—DECEMBER 1986
Includes listing from the following articles, and more.
September Issue #119
"Algorithms: Curve Fitting with Cubic Splines" by Ian E. Ashdown.
October Issue #120
"C Chest: More, a File-Browsing Utility" by Allen Holub. "Processors: TNZ:
November Issue #121
"Digital Dissolve" by Mike Morton.
December Issue #122

Dr. Dobb's Listings #3/86 **Item #172 $14.95**

Please specify MS-DOS, Macintosh, or CP/M. For CP/M disks specify: Apple, Osborne, Kaypro, Zenith Z-100 DS/DD, 8" SS/SD (1986 Listings Only).

Order Form

ORDER NOW! ORDER NOW! ORDER NOW! ORDER NOW! ORDER NOW! ORDER NOW!

NAME ______________________________

(Please use street address, not P.O. Box)

ADDRESS ______________________________

CITY ______________ STATE __________ ZIP __________

DAY PHONE ______________________________

Return form
OR

CALL
800-533-4372
TOLL FREE

In CA call
800-356-2002

☐ Check enclosed. Make payable to:
M&T Publishing, Inc., 501 Galveston Dr., Redwood City, CA 94063

☐ Charge My:

☐ Visa ☐ MasterCard ☐ American Express.

Name on card ______________________________

Account No. ______________ Expiration Date __________

Signature ______________________________

For disk orders, please indicate format. Refer to ad for standard format availability for each product.

☐ **MS/DOS**	☐ **Amiga**	**CP/M**	
☐ **Macintosh**	☐ **Atari 520st**	☐ Kaypro	☐ Osborne
☐ **Apple II**		☐ 8″ SS/SD	☐ Apple
		☐ Zenith Z-100 DS/DD	

QUANTITY	ITEM #	DESCRIPTION	UNIT PRICE	TOTAL PRICE

SUB-TOTAL __________

CA residents must add applicable sale tax on merchandise total ________ %

SALES TAX __________

Shipping must be included with order. See rates below.

SHIPPING __________

TOTAL ORDER __________

In U.S. For Bound Volumes, add $3.25 per book. Add $9.25 for Special C Packages. For other books and disks, add $2.25 per item. *Outside U.S.* For Bound Volumes, add $6.25 per book surface mail. Add $18 surface mail for Special C Packages. For other books and disks, add $5.25 per item surface mail. Foreign airmail rates available on request. For Fast Service and reduced shipping costs, Germans may order direct from: Markt & Technik, Buchverlag, Ilans-Pinsel-Strasse 2, 8013 Haar bei München. Call Germany 89-4613-221 for prices in Deutsch Marks.

3126S

M&T Books & Software

ORDER FORM

Please fold along fold-line and staple or tape closed.

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

First Class Permit No. 790 Redwood City, CA

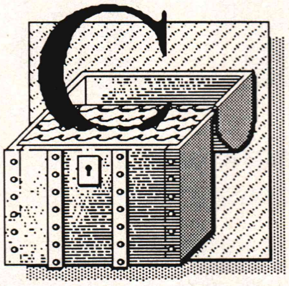
Postage Will Be Paid By Addressee

M&T Books & Software Tools

501 GALVESTON DRIVE
REDWOOD CITY, CA 94063

Please fold along fold-line and staple or tape closed.

Nr: A C Implementation of Nroff, Part 3



This is the continuation of the users' manual for nr, my text formatting program. On the source code disk I show how the commands are used by presenting an implementation of the ms macro package. (See the end of this column for information about the source-code disk.)

Tabs and Leaders

Nr supports arbitrary tab stops that can be placed at any column. Tabs are represented in the text with either an ASCII TAB character (Ctrl-I) or with the *\T* escape sequence. Tabs are expanded at input time to the number of space characters needed to get to the indicated column. They don't work very well in proportionally spaced fonts for this reason. Three types of tabs are supported: left adjusting, centering, and right adjusting. The left-adjusting tabs are the normal variety—the text following the Ctrl-I or *\T* is aligned with the next tab stop. Centering and right-adjusting tabs are more complicated. If a tab stop is a centering one, then all text between the next two Ctrl-Is is centered on the next tab stop. Similarly, right-adjusting tabs cause the text to be right-adjusted on the next tab stop. For example, the following two commands clear all the default

by Allen Holub

tab stops and then set up a left-adjusting, centering, and right-adjusting tab in columns 10, 20, and 30:

```
.ta
.ta 10L,20C,30R
```

Given the input:

```
\T¦\T¦\T¦\T
\Tleft\Tcentered\Tright\T
```

the following will be printed:

```
¦              ¦              ¦
left       centered       right
```

The vertical bars mark the tab-stop positions.

Leaders are like tabs except that the leader character rather than the tab character is used to pad out the text. The default leader character is a period. You use leaders for things such as tables of contents in which you want a string of dots between the last word in the chapter title and the page number. A leader is signaled by embedding an ASCII Ctrl-A in the text (or by using the *\A* escape sequence). For example, the following sets up a tab stop at column 20 and then prints a table of contents entry with an intervening leader:

```
.ta
.ta 20
Chapter 2 \A 17
```

The foregoing will print as:

```
Chapter 2 . . . . . . . . . . 17
```

.ta [A,B, . . . Z]—the argument, if present, is a comma-delimited list of tab stops. Each element of the list can be a specific column, as in *.ta 9,17,25,33*, or an offset from the previous number, as in *.ta 8, +8, +8, +8*. In addition, each number can be followed by one of the following tab types: *R*, for right-justified; *C*, for centered; and *L*, for left-justified. An example was given earlier. If no tab type is specified, *L* is assumed. If no argument is given, all tab stops are cleared.

.tp—print out all the current tab stops in a graphical form:

. . . . *L* . . . . *C* . . . . *R* . . . . *L* . .

where the tab positions are marked with an *L*, *C*, or *R* depending on the tab type.

.tc C—set the tab-expansion character to *C*. The default tab-expansion character is a space. If no *C* is given, tab expansion is disabled.

.lc C—change the leader character from a period to *C*.

Control Flow

Though nr doesn't support a fancy control-flow language, it does support *if* and *if . . . else* mechanisms. The control-flow statements nest. The expression syntax described earlier is also used in an *if* statement, so all the expression operators are available to you here. An expression that evaluates to zero is false; nonzero expressions are true. The basic form of the *if* statement is *if expr action*, where *expr* is any expression involving constants, number registers, and so forth, and *action* is any single dot command or text. For example, in:

```
.if  "\n% != 1"  .bp
```

the *.bp* will be executed only if you're not on page 1. In

```
.if  "\n% == 1"  This is the Title
```

the text *This is the Title* is printed only on page 1. You could also use the *if . . . else* form of the command:

C CHEST
(continued from page 130)

```
.ie "\n% != 1" .bp
.el This is the Title
```

You can combine several statements into a block using the two block commands (*.{* and *.}*). For example, in:

```
.ie "\n% != 1" .{
.     bp
.     sp 10
.}
.el This is the Title
```

both *.bp* and *.sp 10* will be executed if you're not on page 1. For nroff-compatibility reasons, you can also use the less readable:

```
.ie "\n% != 1" \{\
.bp
.sp 10 \}
.el This is the Title
```

if you like.

.if condition—a simple *if* statement (that doesn't take an *else* clause). The expression parser described earlier is used to evaluate *condition*. Two special *conditions* are supported:

.if e action
.if o action

The *e* evaluates to true if the current page number is even; the *o* is true if you're doing an odd page.

.ie condition—the *if* part of an *if . . . else*. It is otherwise used like an *.if*.

.el—the *else* clause part of the *.ie* command.

.{—start a block for an *.if*, *.ie*, or *.el*. The \{\ and \{ escape sequences are mapped to a *.{* for nroff compatibility.

.}—terminate a *.{* block. The \} escape sequence is mapped to a *.}* for nroff compatibility.

Hyphenation

Nr supports automatic hyphenation, enabled with a *.hy* command and disabled with a *.nh* command. The nroff *.hy* command takes arguments, but the nr variant ignores its arguments.

A conservative hyphenation algorithm is used to avoid incorrect hyphens. In addition, only words composed of lowercase alphabetic characters are hyphenated. If the word contains a hyphen, it is always subject to being broken at the explicit hyphen. In general, nr won't hyphenate a word if it's not sure what to do. Nonetheless, it does make occasional mistakes. You can put a soft hyphen into the word to tell the program where a hyphen can go—the \% escape sequence is a soft hyphen. For example, nr will not hyphenate *hyphenate* correctly (it will try to make it *hyphe-nate*). You can correct this with *hyphen\%ate*. The \% is ignored if no hyphen is inserted. If \% precedes the word, that word won't be hyphenated. If a word contains a soft hyphen, nr will not rehyphenate it.

.hy [N]—enable hyphenation; *N* is ignored.

.nh—turn off hyphenation (turned on with a *.hy* command).

Three-Part Titles

Three commands are used to support

three-part titles:

.tl /A/B/C/—print a three-part title. *A*, *B*, and *C* are strings, where *A* is left-justified, *B* is centered, and *C* is right-justified on the page. Any of these can be omitted, as in:

```
.tl ///page %/
.tl //−%−//
```

The first character in the string (a / here) is used as a delimiter and can be any character. The title is printed at the current page offset but indent is ignored. The title width is defined with the *.lt* command. A % character is special in a title. It is replaced by the current page number, printed in the current format associated with the % number register. For example, you can produce Roman-numeral page numbers with:

```
.af % i
.tl //−%−//
```

The *.tl* command is usually used in a top-of-page or bottom-of-page macro.

.lt [+−]N—specify the width of a three-part title, in spaces. Because this command does not affect the *.ll* command, it's possible to have a title with a different width from that of the body of the text.

.pc C—change the character used to indicate a page number in a three-part title from % to *C*. This is useful if you want to put a percent sign in the title itself.

Output Line Numbering

```
   Nr can automatically number out-
   put lines for you—in fact, I use it to
   generate all the numbered listings
 5:for C Chest. You enable output line
   numbering with a .nm command.
   Note that this command behaves a
   little differently from the nroff
   equivalent, mostly because I can't
   figure out how the nroff one
10:works. Numbers are printed right-
   justified in a three-space-wide
   field. Syntax is .nm N M S, where
   N is the number used for the first
   line, M is a line number multiplier
15:and S is a string that's printed to
   the right of each number. The
   number is printed only when the
   current output line number is an
   integer multiple of M. When it's
20:not, a filler composed of three
   space characters is printed instead
   of the number. This paragraph
```

was output with *.nm 1 5*:

.nm N M S—enable or disable line numbering. If you need to change *M* without changing *N*, use *.nm x M S*, where *x* is any nonnumber. The same goes for *.nm x x S*. You can turn off line numbering by issuing a *.nm* with no arguments. If you want to reenable it without resetting the line number, use *.nm x*, where *x* is any nondigit. In addition, the line number used by *.nm* is stored in the predefined *nl* number register.

Nr provides several mechanisms other than the command line for getting input or sending output.

.nb [args]—enable or disable blank-line numbering. Usually blank lines are not numbered—they are output but no number is printed and the output line number is not incremented. A *.nb X* causes blank lines to be numbered too. A *.nb* with no arguments disables blank-line numbering. This command is not supported by nroff.

Input and Output

Nr povides several mechanisms other than the command line for getting input or sending output to the file or the console:

.cf file—copy *file* directly to standard output without any sort of processing. This command is useful for automatically downloading fonts.

.tm string—print *string* directly to standard error. One of the uses for this command is to print diagnostics. My version of the ms package, for example, prints the page number at the top of each page so you can see

C CHEST

(continued from page 133)

where you are in the formatting process, even if output's redirected. You have to use *\N* to get a new line into a *.tm* string (as in *.tm page \n%\N*).

.mf macro file—copy the contents of the named macro into the indicated file. This is not an nroff command. It's useful primarily for indexes. You can write a macro for collecting index entries, and this macro would automatically append some sort of reference information and the current page number to a special macro every time that it was called. After the entire document has been processed, the macro could then be transferred to a file so that you could modify the data.

.ou string—send *string* directly to the current output, without going through the normal text-processing mechanism. This also is not an nroff command. This command is for sending control sequences directly to the printer—that is, for initializations and so on. Use *\x<2 hex digits>* to send nonprinting characters. Note that the *−c* flag (which causes control characters to be printed in readable form) affects the output from this command.

.rd [prompt]—read insertion from standard input rather than from the current input file. Reading stops when two new lines in a row are encountered. This command allows you to insert text interactively into a document. The *prompt*, if any, is printed (and the bell is rung) before any text is read.

.so file—get (source) input from the named file. The position in the current file is remembered, and processing will continue when the source file is exhausted. This command works like an *#include* directive in C does. The *.so* command is replaced by the contents of the indicated file.

Miscellaneous

.\"—signifies a comment. The entire line is ignored. Note that a dot on a line by itself is also considered to be a comment line.

.db [1]—enable or disable debugging mode. A *.db x* enables debugging mode (same as $-v$ $-c$ on the command line), and a *.db* without an argument disables debugging mode. This is not an nroff command.

.ex—exit back to the operating system just as if input had ended. The end macro is executed.

.ig [xx]—ignore all input until a line starting with *.xx* is found, where *xx* is the argument to *.ig*. In the absence of *xx*, . . is used.

.mc string [N]—specify a right margin character, and print *string*, *N* spaces to the right of the current right margin. This usage differs from nroff, which uses a single character rather than a string. If no arguments are present, the margin character is disabled. The string is limited to 20 characters (including any spaces implied by *N*). If *N* is missing or 0, 2 is used.

.ml string—print the string at the left margin rather than the right margin (it works like *.mc* does). The page offset must be at least as large as the string, which is limited to 21 characters. This command is not supported by nroff.

.wa [N]—wait for about *N* seconds (at most $N + 1$). If *N* is 0 or if no argument is given, a prompt is printed and the program waits for you to type Enter. This command is not supported by nroff.

.ws N—enable WordStar-mode output. If *N* is 0 (or missing), WordStar mode is disabled. If *N* is 1, all single new lines are mapped to WordStar soft carriage returns. Note that double new lines, as are used to create a blank line, map to two hard carriage returns. This way you can get a hard carriage return at the end of a paragraph by putting a blank line after every paragraph. An *N* of 2 is handled like $N=1$ except that single carriage returns are replaced with space characters rather than soft carriage returns. Note that you'll also want to do the following:

```
.po 0   \" No page offset
.bd \x02 \x02   \" ^B for bold
.ud \x13 \x13   \" ^S for underline
.od \x18 \x18   \" ^X for overstrike
```

C CHEST

(continued from page 135)

This command is not supported by nroff.

Escape Sequences

Escape sequences are special sequences of characters that either tell nr to do something or signal some sort of macro expansion (that is, the escape sequence will be replaced by some other text). They are all introduced with a leading escape character. This character is a backslash by default, but you can change it with the *.ec C* command, where *C* is the new escape character.

Nr expands escape sequences in three distinct modes: normal mode, copy mode, and nroff copy mode. In normal mode, usually in effect, all escape sequences are expanded. Copy mode becomes active when a macro is being defined—that is, at definition time, the contents of the macro or string are copied into the macro in copy mode. The macro is expanded in normal mode, however. In copy mode only two escape sequences are recognized: \", which introduces a comment; and *\(CR)*, a backslash at the end of the line, which is a hidden line feed (that is, the current line is merged with the next line). All other escape sequences are just copied into the macro intact. The real nroff has a less restricted copy mode in which all the following are recognized:

```
\(CR) \" \. \' \$N \nx\n(xx \n+x
                    \n+(xx \*x \*(xx \\
```

Usually this is more trouble than it's worth because you have to use \\ every time you want to get a \ into a macro (to prevent it from being expanded at definition time). You can use nroff copy mode instead of normal copy mode, however. It is enabled with a *.cm 1* command and disabled using *.cm* without the argument. Supported escape sequences are summarized in Table 1, page 137, and are described in depth in the following paragraphs.

\"—introduces a comment. All characters following the \" on the line and all white space preceding the \" on the line are ignored. A \" at the beginning of line is treated as if it were a blank line (the fill buffer is flushed and a blank line is written to the out put). The .\" command, however, causes the line to be ignored entirely (for example, no blank line is printed).

\(CR)—ignores the end of line—that is, all text on the following line is merged with text on the current line, as if the two lines were one.

\.—a dot that's never interpreted as a command character.

Nr expands escape sequences in normal mode, copy mode, and nroff copy mode.

\'—a backquote that's never interpreted as a secondary command character.

\\—expands to a backslash.

\$N—a macro argument, where *N* is a number in the range 1−9. A macro defined with:

```
.de XX
arg 1 <\$1>
arg 2 <\$2>
arg 3 <\$3>
..
```

and invoked with:

```
.XX "this is one argument" doo wha
```

will print:

```
arg 1 <this is one argument>
arg 2 <doo>
arg 3 <wha>
```

**x*—expands to the contents of a string named *x*. Strings are created with a *\.ds* or *\.as* command. Note that the string could also be expanded as if it were a macro (using *\.x*). Nested * expansions are supported, and the strings can contain other escape sequences (such as *\nx*).

*(*xx*—expands to the contents of a *x* string having the two-character name *xx* (works like *x does).

\nx, *\n(xx*, *\n+x*, *\n+(xx*—interpolates number registers, discussed earlier.

\%—indicates a soft hyphen. Soft hyphens are used to indicate places where a word may be hyphenated. If hyphenation is disabled or if the word isn't at the end of a line, then the soft hyphen is ignored. For example, *hy\%phen\%ate* tells nr to hyphenate *hyphenate* either after *y* or *n*. Words with soft hyphens in them will not be rehyphenated by the automatic-hyphenation algorithm. You can prevent a word from being hyphenated by preceding the first letter with a soft hyphen.

\&*c*—treats the *c* as a literal character. Note that this is different from the normal nroff syntax, which treats \& as a zero-width, nonprinting character. Because nr uses several characters whose values are greater than *0x7f* internally, \& is the only safe way to get such a character through to the printer. For example, if you need to get a *0x8a* to the printer without nroff intercepting it, use \&*x8a*. The character following the \& can be any single character or escape sequence that evaluates to a single character.

\(SP) (a backslash followed by a space)—a nonpaddable and nonbreaking space. Given two words separated with a nonpadding space (*word\ word*), the justification algorithm will never add additional spaces between the words and the two words will always be on the same output line.

\−—evaluates to a dash (it's a minus sign in troff).

\|, \^, \0—nr ignores the first two and maps a \0 to a normal space character. These sequences are supported for compatibility with troff, which treats \| as a thin space, \^ as a somewhat thicker but nonetheless thin space, and \0 as a digit-width space.

\A—a visible leader character. It's the same as a Ctrl-A embedded in the

text.

\L'Nc', *\l'Nc'*—the two line-drawing functions. *\L'Nc'* evaluates to a vertical line composed of *N c*s stacked one on top of the other. For example, the command *\L'3+'* prints:

```
+
+
+
```

The cursor is positioned immediately below the bottom plus sign (you can use *\v* [discussed later] to get back to the top). If no character is specified explicitly, a vertical bar is used: *\L'3'* prints a three-line-high bar.

The *\l'Nc'* works just like *\L* does except that a horizontal line is drawn. For example, *\l'20−'* draws a horizontal line composed of 20 dashes. The default character is an underscore. Note that you can't use an escape sequence for the line character (as in *\l'10\x85'*). You can define a string to do this though:

```
.ds li \\l'10\x85'
\*(li
```

\N—a new line that can be embedded in a string or *.tm* command.

\T—a visible tab character, it is replaced with a Ctrl-I, which will be expanded as the input is processed.

\a—a nonexpanded leader character. That is, it's a Ctrl-A that will make it through to the output without being transformed into a series of dots or whatever.

\d—sends the cursor down half a line. N^2 can be done with *N\u2\d*.

\e—a printing version of the current escape character (the one that was active when the *\e* was encountered in the input). This is more convenient than \\ because many macros will create other macros on the fly, and each level of secondary macro will also expand backslashes. Sometimes it can take as many as six or eight backslashes for one to make it all the way to the output. A single *\e*, however, is never interpreted as a backslash—it always gets to the output unmolested.

\fF—changes fonts. For example, the word *italics* was created with *\fIitalics\fP*. (See the description of the *.ft* command for more information.)

\h'N', *\h'Nu'*, *\v'N'*, *\v'Nu'*—give you fine control over cursor motion. The *\h* command is for horizontal motion, and the *\v* is for vertical. *N* is the number of lines or spaces, as appropriate. *N* can be negative. For example:

X\h'4'X\h'−1'\v'2'X\h'−5'X

prints:

```
X    X
X    X
```

It can be broken up into:

X	print an *X*
\h'4'	move four spaces to the right
X	print another *X*
\h'−1'	back up one position (over the last X)
v'2'	go down two lines
X	print a third *X*
\h'−5'	back up five spaces
X	print the last *X*

If the character *u* follows the count, then motion will be in terms of vertical and horizontal units, as defined with the *.vd* and *.hd* commands, instead of lines and spaces.

\o'ab'—superimposes all characters between the quotes one on top of the other. For example, *\o'0/'* can be used to print Ø. Use the special escape sequence \' to put a single quote into the list.

Copy mode	
\"	Comment (deletes all following text and all preceding white space).
\(CR)	Ignore the end of line.
Expanded in nroff copy mode but not in normal copy mode	
\\.	A dot that's never interpreted as a command character.
\\'	A backquote that's never interpreted as a command character.
\\\\	A backslash.
\$N	Macro argument, where 1 <= *N* <= 9.
*x	String *x*. Nested * expansions are supported, and the strings can contain other escape sequences (such as *\nx*).
*(xx	String *xx*.
\nx	Number register *x*.
\n(xx	Number register *xx*.
\n+x	Number register *x* with auto preincrement.
\n+(xx	Number register *xx* with auto preincrement.
Expanded only when not in either normal or nroff copy mode	
\%	Soft hyphen.
\&c	*c* is literal (can be *\xDD*) (nonstandard).
\(SP)	Nonpaddable nonbreaking space.
\−	Dash (minus sign in troff).
\0	Normal space (digit-width space in troff).
\A	Same as ^*A*.
\L'Nc'	Vertical line of *N c*s (default *c* is *¦*).
\l'Nc'	Horizontal line of *N c*s (default *c* is underscore).
\N	New line that can be embedded in a string or *.tm* command.
\T	Same as ^I.
\X	Where *X* is any other character, that character.
\a	Nonexpanded leader character.
\d	Down a half line.
\e	Printable version of current escape character.
\fF	Change to font *F*.
\h'N'	Horizontal motion by *N* spaces (*N* can be negative).
\h'Nu'	Horizontal motion by *N* units (as defined with *.hd*).
\v'N'	Vertical motion by *N* spaces (*N* can be negative).
\v'Nu'	Vertical motion by *N* units (as defined with *.hd*).
\o'ab'	Superimpose all characters between the quotes (overstrike).
\r	Up one line.
\t	Nonexpanded tab character.
\u	Up a half line.
\xDD	Where *DD* is two hex digits, that character.
\zc	*c* is zero width.
\{	Start block (see also *.{*).
\¦	Ignored (thin space with troff).
\}	End block (see also *.}*).

Table 1: *Nr-supported escape sequences*

Flotsam and Jetsam

Declarations and Definitions in One File

The matter of declarations and definitions that I discussed in last month's Flotsam and Jetsam can cause maintenance problems. You can define (allocate space for) a variable in only one place in your program. Nonetheless, you have to declare the variables (with *extern* statements) in every file that uses the variable.

Michael Yam of NYC suggests a solution to this problem: "Managing globals can be messy, particularly in a C program that has many modules. One of the more difficult tasks in a large program is coordinating the variable declarations (the *extern* statements in a .H file) with the definitions (where the space is allocated in a .C file). You can both define and declare all globals in one place by using the C preprocessor, however, thereby making globals easy to track and document. Let's say you have three modules: testmain.c, test1.c, and test2.c. The *main( )* subroutine is in testmain.c, which also includes the following statements:

```
#defineALLOCATE
#include "testmain.h"
```

Other files may include testmain.h, but none of these other files includes the *#define ALLOCATE*—that's in testmain.c. Testmain.h holds all global definitions and declarations and looks like this:

```
#ifdef ALLOCATE
#define GLOBAL
#else
#define GLOBAL extern
#endif
GLOBAL int   glob1;
GLOBAL int   glob2;

GLOBAL struct
{
   int x, y, z;
}
world;
```

When you compile testmain.c, *GLOBAL* expands to nothing and the variables are defined (space is allocated for them). In all other modules, because *ALLOCATE* isn't *#defined*, *GLOBAL* expands to the keyword *extern* and variables are declared.

"I don't think this approach is anything new, but so few programs take advantage of this technique."

Michael's correct in thinking that the technique's not new, but it's certainly useful at times. When you use it, however, be careful of static initializers, which can't be used in *extern* statements. A good solution is to use the variant of the *D( )* macro I discussed a few months ago:

```
#ifdef ALLOCATE
#define INIT(x) ={ x }
#define GLOBAL
#else
#define INIT(x)
#define GLOBAL extern
#endif

GLOBAL int x[ ] INIT( 1, 2, 3, 4 );
GLOBAL char *y[ ] INIT( "quick",
                   "brown", "fox" );
```

As before, if *ALLOCATE* isn't *#defined* then the initializations aren't compiled. When *ALLOCATE* is *#defined*,

```
int    x[ ]  ={ 1, 2, 3, 4 };
char *y[ ] ={ "quick", "brown", "fox"
};
```

You may need two *INIT* definitions because some compilers won't accept curly braces around single objects, as in:

```
int   z = {1} ;
```

Use:

```
#define INIT2(x) = x
```

Finally you can use a general-purpose initialization macro that includes all the brackets and equal signs with it:

```
#if def ALLOCATE
#define INIT(x) x
#else
#define INIT
#endif
```

You can then say:

```
GLOBAL x[ ] INIT(={1,2,3});
```

C CHEST
(continued from page 137)

r—sends the cursor up one line.

t—a nonexpanded tab character (Ctrl-I). Like a *a*, it will make it all the way to the output without being expanded by the tab-processing routines.

u—sends the cursor up half a line.

xDD—gets binary information to the printer. *DD* is two hex digits (two are required). For example, an ASCII escape character can be sent to the printer with a *x1b*.

zc—says that *c* is a zero-width character. For example, â can be printed with *z^a*.

{, *}* —form a block in an *.if*, *.ie*, or *.el*. For example, in:

```
.if( \nx ) \{\
.in +10
.ti -10 \}
```

both the *.in* and *.ti* are executed if *nx* contains a nonzero number. The *{* and *}* are supported primarily for nroff compatibility. The nr commands *.{* and *.}* tend to be more readable:

```
.if( \nx ) .{
.in +10
.ti -10
.}
```

Availability

The February, March, and April 1987 C Chests have been combined in *Nr: An Nroff-Like Text Processor for MS-DOS*. This reprint is available with a source-code disk for $29.95. Send prepaid orders to M&T Books, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600, extension 216. Please add $2.25 for shipping and handling ($5 for foreign orders).

Missing Subroutines

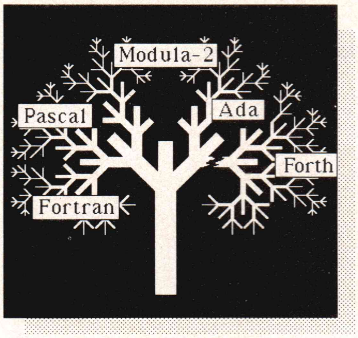
The subroutines *newsample()*, *running_mean()*, and *deviation()* were referenced in February but will be published in the May listings. The *ferr()* subroutine was referenced in February and published in March (page 48). The *err()* subroutine is just *ferr()* without the *exit()* call.

DDJ

(Listings begin on page 84.)

Vote for your favorite feature/article.
Circle Reader Service **No. 5.**

People in Programming



Programming begins with people and their problems: a good program is a sound solution that the user likes. The initial problem in programming is not communicating with the machine but with the client. Discovering what the client wants and—more important—what the client needs, helping the client understand trade-offs and alternatives, and in general learning enough about the client and his or her relationship with the problem so that the program is born with the greatest chance of success—all this requires techniques that do not fold neatly into algorithms.

Programmers sometimes come to grief not because of their lack of technical skills with the hardware and the software but because of misunderstandings and imperfect compromises with the people involved.

You might be interested in two books that seem particularly interesting and helpful in extending skills in the people direction. The first is *Getting to Yes*, by Roger Fisher and William Ury (Boston: Houghton Mifflin, 1981). This is the only book on negotiation I have seen that describes a method instead of offering only a potpourri of unrelated tactics. Not only does it provide a method but it also gives a rationale that demonstrates both the effectiveness and the legitimacy of the techniques.

Negotiation is a primary people skill in programming. Who has not encountered a client whose desires exceed the budget or whose whims

by Michael Ham

overburden the hardware? The programmer often must tactfully devise and suggest practical alternatives to wishful thinking and then negotiate for their acceptance. The unprofessional programmer's approach is either to do slavishly whatever the client asks, however inefficient or dubious, or to implement the programmer's own ideas, arrogantly ignoring the wishes of the client. The professional, on the other hand, recognizes the importance of educating the client and negotiating a sound solution.

Besides negotiating on the technical aspects, programmers (particularly freelance programmers, whose every job involves a contract) must be competent at blunt business negotiation. *Getting to Yes* is as close to a complete manual of negotiation as you can get, with application in every area of negotiating.

The key to successful client relations involves more than a sound technical solution in the context of a well-negotiated agreement. The relationship is woven from a myriad of daily exchanges, with the resulting fabric called cooperation. *The Evolution of Cooperation*, by Robert Axelrod (New York: Basic Books, 1984) provides a computer context for this elemental mode of human exchange.

The book describes two tournaments in which programs were developed to play out strategies of cooperation (or noncooperation) in a prisoners' dilemma situation: in which mutual betrayal gets nothing, but one betrayal gains significant advantage over mutual cooperation and being betrayed gets nothing or little. The explorations are intriguing. You cannot push too far the social analogies of computer strategies, but playing with the ideas is at least stimulating, especially since the strategy that won the two tournaments has a long and honorable history.

Number Input

In my inaugural column (July 1986), I proposed a number-input word to collect numbers calculator-style, displaying appropriate punctuation, tolerating normal user errors, and giving the programmer control over the number of digits that can be entered before and after the decimal point.

Other languages (notably C) come provided with a toolbox already stocked with useful routines such as number input. Forth instead offers to its programmers the elements from which they can build tools. Over time, Forth programmers accumulate a collection of handmade tools, often beautifully worked to fit precisely the special nature of their particular applications. Moreover, because he or she built the tool, the programmer understands exactly not only what it does but also how it does it—the limitations and the strengths—and can readily modify the tool to fit any mutation of the original situation.

The drawback to Forth's approach is that building and polishing the tools take time. Sometimes it would be nice to take a close-enough solution off the shelf instead of fashioning the perfect fit to the specific problem. One source of such solutions is the set of modules written in other languages—C or FORTRAN, for example—that have a store of existing tools. Laboratory Microsystems has recently published a Forth, which it calls UR/Forth, that can be linked to such alien modules.

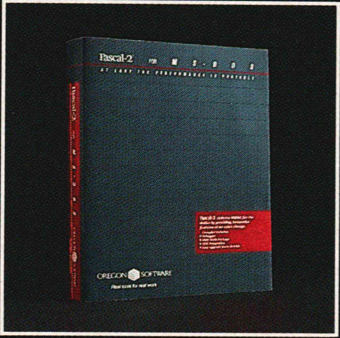
Another approach is to augment your collection of software tools by reading (and adapting) published tools. Here I discuss my current solution to the common problem of collecting a number from the key-

TALK OF THE TOWN

One language supports this community. That language is Pascal-2, now on the PC and producing the fastest, most compact code available. For the professional programmer, imagine what you can do with this power:
- Cut execution time by 20% to 200%
- Transport MS-DOS programs to VAX, PDP-11, and 68000 machines with only minor adjustments
- Cut executable program size by up to 50%
- Use all of DOS-addressable memory through efficient large-memory model
- Speed error correction and save development turn-around time with sophisticated error checking and reporting
- Find and fix logical errors with the interactive source-level debugger
- Access DOS services and network files
- Call Microsoft FORTRAN, C, Pascal, and assembler
- Upgrade from TURBO Pascal with compatible strings, equivalent procedures and access to TURBO graphics.

Pascal-2™

FOR MS-DOS



Plus!
- *Intel CEL87 mathematical library for scientific computing*
- *A special interface between Pascal-2 and the programmable BRIEF text editor (editor optional).*
- *Certified ISO standard Level 1.*

Dramatically improve your productivity and introduce your PC software to the VAX next door.

Call or write OREGON SOFTWARE, INC.
6915 SW Macadam Avenue,
Portland, OR 97219 (800) 367-2202
TWX: 910-464-4779 FAX: (503) 245-8449

OREGON SOFTWARE

Real tools for real work

AT LAST THE PERFORMANCE IS PORTABLE

The following are trademarks: Oregon Software, Pascal-2, Oregon Software, Inc.; IBM, PC-AT, PC-DOS International Business Machines Corporation; Intel, Intel Corporation; MS, Microsoft Corp.; TURBO Pascal, Borland International, Inc.; BRIEF, UnderWare Corp.; PDP, VAX, Digital Equipment Corp.

Circle **no. 357** on reader service card.

STRUCTURED PROGRAMMING

(continued from page 140)

board—the word *DIGITS* (see Listing One, page 118). This word leaves two numbers on the stack—the number entered (as a signed double-precision number) and above it the count of the digits entered. Decimal fractions are scaled to integers, and the decimal display is merely cosmetic—for example, the number displayed as 0.0023 is in fact stored as a double-precision 23.

The count of digits distinguishes an entered 0 from the 0 that results from no entry. "No entry" results from the user pressing Return when the entry field is blank. I designed "no entry" to leave a double-precision 0 on the stack because I like my general-use tools to leave the same number of arguments under all conditions.

The count of digits has some other uses. If the count is less than 5, for example, the entered number is less than 10,000 and thus the top cell of the double-precision number will be 0, which can be dropped to leave a single-precision version of the entry.

Most of the complexities in the word are because of various special cases. Each of the following revealed a bug during the development of *DIGITS*:

- pressing a clear-entry key (C or B or the space bar) when the entry is already clear
- starting with an old number equal to 0
- having fewer nonzero digits in the number than the number of decimal places entered

When you start with an old number of 0, the single 0 digit must not be counted as you begin to enter numbers. If it is counted, you will be able to enter at most one fewer digits than you should because the keystroke count routine will be initialized to count what is in effect a leading 0: the 0 that was the old number.

The number of nonzero digits in the number can be less than the number of decimal places entered (for example, 0.0023 has four decimal places entered but is represented as the two-digit number 23). The difference between the number of digits in the number and the number of digits to the right of the decimal can thus be negative. It is for this reason that you see *0 MAX* in computing the number of commas.

Another minor complexity results from the incomplete complement of double-precision operators provided by most Forths. My January column suggested a naming scheme for arithmetic operators. In terms of those names, this application could use the two operators *M*D* (a double and a single factor, single on top, giving a double product) and *M/D* (a double dividend, a single divisor, and a double quotient). The pair *D** (two doubles as factors with a double product) and *D/* (a double dividend and a double divisor giving a double quotient) would serve equally well.

With *M*D* (or *D**) the routine could accumulate the number as a double: each time a digit is entered, the number so far entered would be multiplied by 10 and the new digit added. With *M/D* (or *D/*) the backspace would be easy to implement by dividing the number so far accumulated by 10 to strip off the last digit entered.

One approach is to write definitions for these operators. I use an alternate route: I accumulate the number as a string of ASCII characters and use *CONVERT* whenever I want the value of the number.

I'll briefly discuss the code, but let me first point out that I do some arithmetic with the flags. Deplorable as the practice may be, I find it irresistible. The important thing for you to know is that these are 83 Standard flags, in which true is shown by -1 (all bits on), unlike the 79 Standard true, which is 1. Thus I use the (83 Standard) flag to decrement or (after negating it) to increment a count. The sign must be reversed if you are using 79 Standard flags.

This code may be more complex than necessary: simplicity is not easily achieved. The code does, however, get the job done, and the response time (on an IBM PC) is totally adequate. Share with me any simplifications you discover.

The first few definitions are tiny tools. You will note that the words to turn the cursor on and off are vendor dependent, and you should check with your own Forth for this type of control. Some Forths automatically extinguish the cursor when *KEY* is ex-

ecuted; this was written in Laboratory Microsystem's PC/Forth, which does not. I use *-CUR* to turn the cursor off, *+CUR* to turn it back on.

The function of the phrase *8 EMIT* is also vendor dependent: some Forths execute a backspace, some display a character. So you may have to revise the definition of *BACK* to make it work as intended: to backspace the cursor as many positions as specified by the number on the stack.

Some definitions, such as *NEW* and *OLD* and *Bs?* and *Cr?*, are nonce words to improve the readability of the code, always an important objective. As usual, I prefer short definitions based on normal English usage. In my eyes, playfulness is more an asset than a detriment, provided that the word's name reflects its effect.

Control of the sound generator is also vendor dependent. In the definition of *BELL*, the stack holds numbers that define the pitch and the duration. I prefer a short beep. The variable *SOUND* provides an easy on/off control for beeping.

Because *PAD* was occupied with other tasks, I created a separate work area for the number being entered. This work area, *#PAD*, will contain the string of ASCII characters that represent the number.

#VAR is an array in which I name each cell using a constant. The constant, returning the address that is its value, acts as a variable name. Some naming conventions are apparent: a name ending with ~ is a Boolean variable; a named prefixed with *ini-tializes an array or variable by setting it to zero. (I read * as "zap.")

The words that collect and edit the character typed have some points of interest. You will note that *FIXUP* converts B or blank to C. C clears the display, and B and blank become synonyms. Also, L and O are converted to 1 and 0, respectively; the user's intention is clear, and overpunctilious programs quickly make enemies.

The word *#?* checks whether the ASCII value is in the range for a decimal digit. *BAD?* leaves a flag (denoted by the suffix *?*) that is true if the entry was bad. Because Forth is not typed, I can use as a flag the value from *#DEC* (the number of places to the right of the decimal). A 0 from *#DEC* (that is, there are no places to the right of the decimal) acts as a false flag; any other

UNIX TOOLS FOR YOUR PC

STRUCTURED PROGRAMMING
(continued from page 143)

value (that is, there are places to the right of the decimal) acts as a true flag. The decimal point is thus allowed as a valid character only if *#DEC* is greater than 0.

The word *#,S* computes the number of commas required, given the number of whole-number digits. If the number of whole-number digits is a multiple of 3, the computed comma count is corrected by decrementing it by 1 (using the 83 Standard true flag of −1). The phrase *0 MAX* corrects for those instances in which the number of nonzero digits entered minus the number of places to the right of the decimal is negative, thus producing a negative quotient.

FULLCNT adjusts the character count of the number of digits and commas to include the decimal point and minus sign (if they are allowed). *BOXSIZE* uses this word to calculate from the number of digits being collected how large a box will be needed: as many spaces as the maximum number of printable characters, plus one space at either end of the box. Because the inverse video on the color screen clips the edges of some characters, I include an extra space at the beginning and end of the number display.

BOXSIZE has one tricky aspect: because I print the leading 0 for decimal fractions less than 1 (for example, 0.0023 instead of .0023), I have to add 1 to the character count if I am collecting such a fraction. I save on the return stack the flag that tells me whether this is that sort of fraction. The flag, after correcting the sign, is added to the count. *BOX* then uses *BOXSIZE* to print an inverse field in which the number will be entered.

One peculiarity in number entry is that some data must be displayed before any number has been entered at all: the minus sign and the decimal point could be the first two keystrokes, and when those are entered, there is still no number to edit. The word −. displays these characters. You will note, by the way, that the minus key works as a toggle so that the minus sign can be entered or altered at any time during number entry.

PUT# prepares the accumulated number for display, leaving on the stack the address and count of the string, which is then displayed in *DISPLAY#*. *?DO* is found in most Forths: it executes the loop only if the two arguments are unequal. If your Forth lacks *?DO*, you can substitute *2DUP = IF 2DROP ELSE DO* for it and follow *LOOP* by *THEN*.

When number entry is complete, the number is left on the stack with any decimal fractions scaled up to an integer value. It is the programmer's responsibility to make sure that the size of the entries allowed (total number of digits, number of decimal places) will result in a number that will fit within the bounds of a double-precision number after scaling.

SCALE# takes care of the situation in which the user did not enter all the fractional digits allowed. If the entry parameters allowed for thousandths (three fractional digits), for example, and the user pressed Return before entering a decimal point (or any digits after it), *SCALE#* will multiply the entered number by 10 three times in order to force the three fractional digits (0s in this case). The word *10D** fakes a double-precision multiply by 10.

The count adjustments in the next group take care of decrements required when backspacing (deleting digits). The words *WHOLE-CK* and *DEC-CK* check the limits imposed by the programmer on the number of digits that can be entered as whole number digits and the number of digits that can be entered as fractional digits. Attempts to transcend the programmer-imposed limit are rejected.

SET-NEG sets the negative variable when an old number is entered. Note that the number is accumulated as a positive number, with the actual sign indicated by the variable *NEG~*.

DSET sets up everything to start the loop and leaves on the stack the loop limits and the number of digits to collect. When you are collecting a new number, the loop limits are m+1 (one more than the number of digits to collect) and 0, but if the routine starts with a number already entered, then the limits are adjusted appropriately. The upper limit is m+1

because the user must be able to enter one more keystroke than the number of digits to collect. Typically that one additional keystroke will be a Return, but it might be a backspace. Any other final keystroke is rejected.

Because the topmost word *DIGITS* is so complex, I wanted to factor out some subroutines to improve readability and comprehensibility (and therefore debugging). But then I had to be able to access the index value from a word outside the loop. The simplest solution was to store the index value into a variable and define the word *"I"* to fetch the variable's value. This would have been a natural place for a *QUAN*, but most Forths don't have them.

You will note that *DIGITS* is a *DO . . . +LOOP* structure. This structure seemed the easiest way to move back (backspacing) and forth (entering a valid character) in the entry or just to remain in place (attempting to enter an invalid character): by having the index increment be negative, positive, or 0, respectively.

The backspace routine, by the way, has to keep track of whether the backspace is over a digit, the decimal point, or the minus sign, or disallowed because no valid character has yet been entered to backspace over. The variety of situations tends to complicate the definition. *BSP-ROU* leaves the appropriate increment to the loop index. When this is the negative of the current index value, the loop returns to its starting point.

DIGITS keeps on the stack the number of digits to be entered because that number is periodically referenced. It would perhaps have been cleaner to park that number inside one of the pseudovariables that make up *#VAR*, but by the time the idea occurred to me, the routine was already working. Once the routine was working, I was disinclined to toy with it.

I suggest that, as an exercise, you modify the routine so that it stashes the number of digits to be entered into an additional cell in *#VAR*, whence it is fetched when needed. When you have the revised routine working once more, you should have a good understanding of this tool, which I hope proves useful to you.

Availability

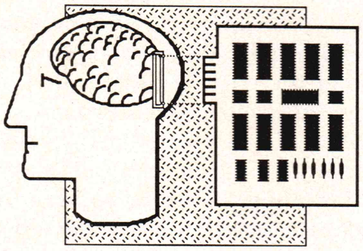
All the source code for articles in this issue (except for C Chest) is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listing begins on page 118.)

Vote for your favorite feature/article.
Circle Reader Service **No. 6**.

Object-Oriented Programming in AI



This month I continue the theme of object-oriented LISPs by focusing on three object-oriented extensions to Common LISP: ObjectLISP, New Flavors, and CommonLoops. Although it is unlikely that any of these precise implementations will become a standard, any such standard will probably combine features of all three.

ObjectLISP

ObjectLISP is the candidate for an object-oriented extension to Common LISP that was offered by LISP Machines. Although it does not now look as though it will provide a substantial part of the standard currently being defined, it is a relatively easy system to understand and has several commendable features. One of the distinctive features of the ObjectLISP approach is eliminating any special syntax for message sending so that object-oriented methods are invoked with essentially the same syntax as any Common LISP function. Another important feature is that ObjectLISP departs from most other object-oriented systems by deliberately making the relation of a class to a subclass the same as the relation of a class to an instance. What this means on the implementation level is that the nesting of closures is used for both specializing and instantiating classes. When combined, as they are in ObjectLISP, these two features result in a simple and streamlined system so that class variables, class functions,

by Ernest R. Tello

instance variables, and instance functions all exhibit the same basic behavior. One of the byproducts of not differentiating between an instance and a class is that, during development, you can use a class as a prototype instance or an instance as a prototype class.

ObjectLISP also has the convenient feature, often not present in object-oriented systems, of allowing the dynamic creation and modification of objects on the fly, as it were, while programs are running. Also, all inheritance operates dynamically. This means that changes in the state of a superclass of an object that are inheritable will do so right when the changes occur.

The basic ObjectLISP system is based on five primitive functions: *make-obj*, *kindof*, *ask*, *have*, and *defobfun*. Creating an object can be as simple as writing:

```
(setq business (make-obj))
```

Frequently, though, the object will be a specialized version of another object that already exists. In this case, an object is created with something such as:

```
(setq wholesaler (kindof business))
```

The *ask* function is used to evaluate a Common LISP expression in a particular object's environment. The *have* function creates variable bindings that are local to objects. These functions are usually used together in ObjectLISP to declare class variables and instance variables. This could be done for the *business* object just created like this:

```
(ask business (have 'type-of-activity
                    'economic))
```

This can then be checked to make sure it has been accepted by the system. If you do so, the terminal screen might read:

```
(ask business type-of-activity)
  economic
```

The *defobfun* function is used to define Common LISP functions that are bound or assigned only to a particular object or class of objects. Continuing with the example I have been using, I might say:

```
(defobfun (calc-net-gain business)
                    (gross-sales costs)
(setq net-gain (- gross-sales total-costs)))
```

The ObjectLISP syntax for calling such a function can be illustrated by:

```
(ask business (calc-net-gain 500 300))
```

Another important capability of ObjectLISP is the ability to create shadowed functions. This is a way in which the inherited functions can be used to create more specific versions of the function for more specialized objects. In many cases, the way this can be done efficiently is by adding only the more specialized parts and then making a call to the inherited function.

Although there is no real difference between an instance and a subclass in ObjectLISP, in practice it is convenient to have a way of using an object as a template for creating other instances of it. One way of doing this is to define an *exist* function for that object as shown in Example 1, page 147. With *exist* functions of this kind, it becomes much easier in ObjectLISP to define instances of objects. So, for example, you could define several business instances as follows:

```
(setq unicomp (kindof business))
(ask unicomp (exist))

(setq softrend (kindof business))
(ask softrend (exist 'ownership-type
                     sole-proprietor))
```

In ObjectLISP, an object is really a list of frames. The first member of the list is its innermost frame, the original bindings supplied to it when it is created. The remaining elements of the list are all the elements that it inherits, appearing in the order in which it inherits them.

Multiple inheritance in ObjectLISP is accomplished by supplying multiple arguments to the *kindof* function. So, for example, if you have also defined an object called *adversary*, then you could define a class called *competitor* using multiple inheritance as follows:

```
(setq competitor (kindof business
                         adversary))
```

At this point, ObjectLISP does not appear to be one of the winning contenders for the standard, partly because it uses dynamic binding but also because it is a new approach that still has not been tried and proven for any appreciable time. Because there are still some difficult and controversial issues in object-oriented LISP, I think that some of the aspects of ObjectLISP, particularly the placing of classes and instances on a common footing and the ability to modify objects on the fly, deserve some serious consideration.

Old and New Flavors

As I said in my last column, the original Symbolics Flavors system was the first commercial object-oriented extension to LISP to gain relatively widespread popularity and to prove the extreme value of object-oriented LISP in practice. The latest software release for the Symbolics 3600 series machines, Genera Release 7.0, now includes New Flavors, the candidate from Symbolics for the object-oriented standard for Common LISP. Symbolics Flavors grew out of the Flavors system developed by the MIT LISP Machine group back in 1979. By 1981, the Symbolics software group had developed a more efficient Flavors system, an object-oriented system that has come to be a favored programming approach both for much of the in-house systems programming at Symbolics and for numerous AI projects carried out by users.

New Flavors represents an attempt to overcome some of the weaknesses encountered by users of the Symbolics Flavors system during the five years of its existence. David A. Moon of Symbolics recently outlined the main goals of New Flavors as follows:

- to encourage greater program modularity
- to facilitate writing large, complex programs
- to provide favorable run-time performance
- to maintain downward compatibility with old Flavors

Like the original Flavors, New Flavors uses the *defflavor*, *defmethod*, and *make-instance* functions for creating objects and procedures. The way the example introduced in the discussion of ObjectLISP would be coded in New Flavors is shown in Example 2, below.

One of the central ideas in New Flavors is the notion of generic functions. The main point of this is to allow distributed definition of functions as well as multiple inheritance of properties. This means both having the same name for a method that varies depending upon the class to which it is bound and being able to use parts of code from various different objects. Toward this end, the *defgeneric* function has been provided.

In New Flavors, generic functions have the same syntax as do nongeneric functions. This has the advantage that any function that is a caller of another function does not need to know which to specify. Other advantages are that all debugging and utility functions designed to work with ordinary Common LISP functions can also work with generics.

New Flavors has adopted a clear set of rules for ordering flavor object components. *Components* are all parts of an object, both those declared directly and those that are inherited. The three rules that are followed are:

- The flavor's own binding always precedes those of its components.
- The local order of components of flavors always adopts the order stipulated in the *defflavor* declarations.
- All duplicate flavors are automatically removed from the sequence.

Method Combination

As I've mentioned before, Flavors

```
(defobfun (exist business) (&rest args &key*
   (name 'no-name-yet)
   (location 'no-location-yet)
   (industry 'no-industry-yet)
   (business-type 'no-bus-type-yet)
   (size 'no-size-yet)
   (year-founded 'no-year-founded-yet)
   (ownership-type 'no-ownership-type-yet)
   (market-share 'no-market-share-yet)
   &allow-other-keys)
   (have 'name name
   'location location
   'industry industry
   'business-type business-type
   'size size
   'year-founded year-founded
   'ownership-type ownership-type
   (apply 'shadowed-exist args)
     ))
```

Example 1: *Defining an* exist *function in ObjectLISP*

```
(defflavor business
 (name location industry business-type size
year-founded
    ownership-type market-share) ()
   :readable-instance-variables
   :writable-instance-variables
   :inittable-instance-variables)

(setq unicomp
 (make-instance 'business
 :name              unicomp
 :location          santa clara
 :industry          computer
 :business-type     software
 :size              18
 :year-founded      1976
 :ownership-type    private
 :market-share      11.3

(defmethod (calc-net-gain business) (gross-
sales costs) (- gross-sales costs))
```

Example 2: *Creating objects and procedures with New Flavors*

(continued from page 147)

was the first object-oriented system to provide the form of abstraction that allowed different parts of the code for functions to be mixed in modular fashion just as complete methods and variables may be inherited. Various built-in combination methods are provided for this purpose. So, for example, programmers can choose between such method-combination modes as:

- calling only the most specific method available in the hierarchy
- calling all the methods in order of specificity, either upward or downward
- trying each method in turn, starting with the most specialized, until one is found that does not return nil

There are also several other built-in combination-method modes.

In addition to defining new methods and selecting built-in combination-method types, programmers can also define new combination methods using the define-method-combination and define-simple-method-combination functions.

Development Tools

New Flavors provides various facilities for inspecting the current state of an object-oriented system under development. You can either invoke them by entering commands or by pointing the mouse at the names of various items on the display. So, for example, you can view either the subclasses or superclasses of a current flavor, and you can view all the instances of a given flavor that are currently alive. These are some of the really useful facilities that an object-oriented system needs if it is to be used for serious AI applications. Despite its lush user environment, some of these features are absent even in Smalltalk.

CommonLoops

The object-oriented extension that has been developed at Xerox PARC has several definite goals and key concepts. It is not surprising that, of all the systems discussed here, it has the most in common with Smalltalk because the amount of expertise present at Xerox with this type of object-oriented system is still considerable. But CommonLoops also represents a departure from Smalltalk in that it offers a clear philosophical vision of how object-oriented programming can be fitted most naturally into the Common LISP dialect and in a way that preserves the greatest amount of generality. It is therefore intended to provide a basis for as many as possible of the serious approaches to object-oriented AI. One of the stated goals of CommonLoops was to provide a general kernel, written in Common LISP, from which any of the major object-oriented systems in use today, such as Flavors, Smalltalk-80, and Loops, could all be implemented. Like Smalltalk, therefore, CommonLoops makes use of the metaclass protocol to implement its class hierarchy system.

The Kernel

The direction taken by CommonLoops is to use an option to the *defstruct* construct in Common LISP in order to define classes. The *:class* option to *defstruct* is employed by CommonLoops to specify the metaclass that will be used in the system to determine how the object-oriented approach to be implemented will behave. The standard metaclasses provided in CommonLoops are *built-in-class, structure-class, list-structure-class,* and *vector-structure-class.*

As does Smalltalk, CommonLoops has various built-in classes. This means that even before a programmer defines any classes, there are already various ones present that describe the behavior of the system. Example 3, below, shows the hierarchy of CommonLoops' built-in classes, shown as they might appear in a "class browser," with the type of class represented in parentheses to the right.

Through this hierarchy of metaclasses, CommonLoops controls the way that the options to *defstruct* determine the form of object-oriented system that will be present. The *structure-class*, for example, is the default class that *defstruct* uses when no *:class* option is specified. The classes that are then created default to a structure that acts like the ordinary *defstruct* in Common LISP. *Abstract-class* is used in the *:class* option for classes that will not themselves be instantiated but act as placeholders in the hierarchy—for example:

```
(defstruct (business (:class
                 list-structure-class)))
```

Table 1, left, gives a list of the main CommonLoops primitives.

Multiple Inheritance

Specifying inheritance from multiple classes is accomplished in CommonLoops through an extension of the *:include* option of *defstruct* to allow it to accept a list of names of classes. Following the same example that I have been using to illustrate multiple inheritance, the *competitor* class, which inherits from the two parent classes *business* and *adversary*, would be implemented in the following way in CommonLoops:

```
(defstruct (competitor (:include
                 business adversary)))
```

```
t         (abstract-class)
 object       (class)
   essential-class          ''
     abstract-class         ''
     built-in-class         ''
     class          ''
     structure-class         ''
     list-structure-class       ''
     vector-structure-class      ''
 number          (abstract-class)
   integer           ''
     fixnum      (built-in-class)
 sequence      (abstract-class)
   list           ''
     cons      (built-in-class)
```

Example 3: *The hierarchy of CommonLoops' built-in*

classes
class-of
defmethod
get-dynamic-slot
get-function
get-slot
mlet
ref
remove-dynamic-slot
remove-method
run-super
specialize
with

Table 1: *Main CommonLoops primitives*

(continued from page 148)

Multimethods

One of the most important innovations in CommonLoops is that of multimethods—procedures that are, in effect, messages sent to any number of objects of different types. So instead of defining the method *draw-at*:

```
(defmethod (rectangle :draw-at) (upper-left-x upper-left-y
        lower-right-x lower-right-y)
   (draw-box upper-left-x upper-left-y lower-right-x
        lower-right-y)
```

as you would in Flavors, in CommonLoops you would write it:

```
(defmeth draw-at
   ((r rectangle) (upper-left-x integer)
        upper-left-y integer)
    (lower-right-x integer) (lower-right-y integer)
    (draw-box upper-left-x upper-left-y lower-right-x
        lower-right-y)
```

In this definition, the first argument to *draw-at* is *r*, which is declared as of the class *rectangle*, and the remaining are screen coordinates, all declared as of the class *integer*.

The implementation of CommonLoops is itself fully object-oriented in the sense that all data structures used to implement the system are objects that are instances of a class. So, for example, when a new method is defined, three new objects are created: the method object, the discriminator, and the discriminating function. The method object is the object that describes the method to be created, and the discriminating function is an object that selects the method that will be called. The discriminator and its own methods use the information in the method object and its own description of a generic function to compile the code for the method. *Generic function* is used here in the same sense as in the discussion of New Flavors.

Method Combination

Method combination is accomplished in CommonLoops using the run-super mechanism. It closely resembles the method combination approach used in Smalltalk, LOOPS, and ObjectLISP. (LOOPS is an AI development tool used at Xerox PARC and will be described in detail in a subsequent column.) The run-super mechanism is implemented using the method and discriminator object described earlier. Because of the use of metaobjects in the implementation of method combination, many interesting research possibilities for AI languages are opened up. For example, through defining specialized method and discriminator objects, a means is available for integrating logic programming into CommonLoops. A prototype for such a system, called CommonLog, has been implemented at Xerox PARC. It is hoped that this will provide the basis for a more advanced AI tool called Vulcan. (This name was apparently not chosen by accident. One of the times I called the Intelligent Systems Lab at Xerox PARC, the entire staff was at the movie theatre to see the debut of *Star Trek IV*).

Future Directions in Object-Oriented LISP

One of the isues still to be settled by the object-oriented LISP community and the object-oriented programming community in general is that of the structural vs. the procedural view of objects. This is the issue of whether the specification or interface description of classes should be purely procedural or split into procedural and structural parts. If purely procedural, an object is defined exclusively by its message protocols. As you have seen, CommonLoops is of the second type because method lookup is achieved by a combination of object structures and discrimination procedures. If things continue to proceed as they have been, it is anticipated that this approach will come to be adopted as the standard one.

On the whole, I don't think it is necessarily a problem of overwhelming difficulty to determine what would be the best kind of standard for the present as an object-oriented extension to Common LISP. Because it is a case of needing some standard now but not having enough experience with this area to fully define the possibilities, the only kind of standard that can at all serve is a partial standard based on those features of the technology that have shown themselves to be the most useful and reliable, while leaving the options as open as possible. To be more specific, I think that a new design for the standard needs to be constructed from the best features of CommonLoops, New Flavors, and ObjectLISP that address as many of the key issues I have been discussing as is feasible.

The main things from ObjectLISP that I feel should not be lost are the ability to modify objects and their variables on the fly and to keep instances and classes on an equal footing. One thing I would particularly like to see is a standard that did not prevent the option of having instantiated objects that were not yet formally members of any class but that at a later time could become "associated" with various classes and gain from what could be inherited from them. Another important issue from the AI perspective is allowing for the coexistence of multiple types of hierarchy in the same binding environment, where the same object can be a member of each of the different hierarchies simultaneously if this is so desired. As explained earlier, this feature appears essential for using objects to develop systems with deep models capable of reasoning about objects in real-world settings in terms of function, location, and generic significance.

In my next column I will continue the discussion of object-oriented programming in AI with a review of PC Scheme—an object-oriented programming system for IBM PCs and compatibles.

Bibliography

Bobrow, D., et al. "Commonloops: Merging Common LISP and Object-Oriented Programming." *OOPSLA '86 Proceedings.*

Dresher, G. "ObjectLISP." *LMI* (1985).

Moon, D. "Object-Oriented Programming with Flavors." *OOPSLA '86 Proceedings.*

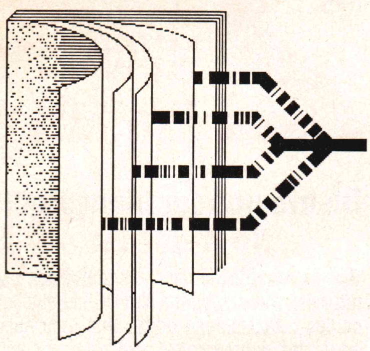
Ressler, J. "Introduction to ObjectLISP." *LMI* (1985).

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 7.**

55
60
50
5
45
40

DDJ ON LINE



The following exchange took place on the message board of DDJ FORUM, our SIG on CompuServe.

#: 7972 S9/UNIX for the PC
17-Nov-86 18:15:23
Sb: Dead Child Floating
Fm: Steve Sampson 75136,626
To: Multitasking expert
I've been trying to get a CRON program running on a Unix clone, but after each task finishes it leaves an entry in the process table—"terminated." Of course, this is unsightly, and I'd like to get it out. Is there a way to pop this without a *wait( )* call? If it will help I can upload the offensive code. Maybe someone has seen this type of behavior before?

***There is a reply: 7986

#: 7986
Sb: Dead Child Floating
Fm: Fred Buck 73327,3604
Is your only beef that the *ps* command shows the process and says it's terminated? For how long a time does this condition continue? If stuff keeps clogging the process table, then that's bad news, but you say it's "unsightly," so I assume your objection is just to the cosmetic aspects. The *ps* command just takes a snapshot of the process table at the time you invoke it, and sometimes the dead stuff just hasn't been cleared out by that time. Do successive *ps* calls keep showing the same terminated process?

***There is a reply: 7995

#: 7995
Sb: Dead Child Floating
Fm: Steve Sampson 75136,626
Yes on the *ps*. I need to find out if the process entry ever goes away, but so far the terminated entry stays around for many minutes. This is just a cosmetic thing and more a question of programming. I'm still coming up to speed, and there's much to discover. I have found that when I kill CRON all the children drop out of the *ps* table.

***There is a reply: 8009

#: 8009
Sb: Dead Child Floating
Fm: Fred Buck 73327,3604
OK, I now believe I understand your situation. Let's see if I can summarize it correctly. You have a single, long-running process that spawns special-purpose children, each of which dies when its special purpose is finished. Such "dead" children show up in the process table as terminated as long as the long-running parent continues to run. You know that a *wait( )* call will clear the entries out of the process table, but you don't want to paralyze your long-running parent by "waiting" for children. If I'm correct about what your problem is, I don't know any way other than a *wait( )* call to do what you want to do, but on the other hand a *wait( )* should be able to be broken by a signal, including an alarm signal. You should be able to clear out the dead children by placing, in some frequently traveled portion of the parent's code, something such as *alarm(1);* followed by *wait( );*. Each time *wait( )* is called, a dead kid will be stripped out of the process table (I assume you don't care about its exit status). If there're no dead kids but one or more alive kids, the *alarm( )* will break out of *wait( )* within one second. If there are no kids at all, *wait( )* should return immediately. I'm assuming you've trapped the alarm signal appropriately. If I'm wrong about this, then just what is your objection to a *wait( )* call?

Oops, *wait( );* should be *wait(0);* or whatever variant of *wait( )* that doesn't care about the child's exit status.

***There are replies: 8020, 8090

#: 8090
Sb: Dead Child Floating
Fm: Steve Sampson 75136,626
Well, I did state the problem in my nonverbose mode, but you summarized it very well. Thanks for the suggested work-arounds; I'll give them a try.

***There is a reply: 8101

#: 8101
Sb: Dead Child Floating
Fm: Fred Buck 73327,3604
I spoke to a friend tonight who has more Unix smarts than I do and can now provide the following additional details: the defunct child processes are more than just cosmetic, they in fact clog up the system process table and can (if sufficiently numerous) inhibit new process creation. In vanilla Unix there isn't really any way to handle such a situation other than the way I suggested or something very much like it. In some versions of Unix (perhaps standard in Berkeley—my friend wasn't off-top-of-head sure), there's a call called *wait3( )*, which can be set to serve the single purpose of disposing of dead children and doesn't wait for a child to die if nothing but living children exist. The reason the dead children disappear when your CRON is killed is that child processes, upon the death of their parent, become adopted children of process 1 (*init*), which is almost always in a *wait( )* and so takes care of them handily and which ignores deaths of children it doesn't remember having spawned itself.

***There is a reply: 8143

#: 8143
Sb: Dead Child Floating
Fm: Steve Sampson 75136,626
Good stuff here, Fred! Appreciate your help on this. I was thinking maybe there was a way to do it but was off on a very different tangent. The info on process 1 adopting children I suspected but didn't know how to get them adopted. Thanks.

#: 8020
Sb: #8009 Dead Child Floating

Fm: Levi Thomas (sysop) 76703,4060
To: Fred Buck 73327,3604
Why do these messages read like a Steven King novel? Dead kids? Paralyzed parents? Macabre metaphors,eh? Shall we start an on-line novel? A two-tiered story—on one level, a technical problem is stated and various solutions suggested. On another level is a horror story in the Edward Gorey style. Sorry, I get carried away sometimes—carry on.
—Levi ("*N* is for Nevil who died of ennui") Thomas

***There are replies: 8027, 8054

#: 8027
Sb: Dead Child Floating
Fm: Fred Buck 73327,3604
Not to mention those instances in which a dead child prevents stuff from being flushed down a pipe. Especially if the child has been spawned by a demon.

***There is a reply: 8029

#: 8029
Sb: Dead Child Floating
Fm: Duane Ellis 76064,1107
If I did not know what you were talking about, you would get some very strange responses from me. As it is, a coworker came by and looked at my screen—I had to explain what is meant by dead children and flushing them down pipes and also the fact that they could have a demon for a parent! Egads!

***There is a reply: 8044

#: 8044
Sb: Dead Child Floating
Fm: Fred Buck 73327,3604
Just imagine if Pat Robertson ever got a hold of this.

#: 8054
Sb: #8020 Dead Child Floating
Fm: Neil J. Rubenking 72267,1531
To: Levi Thomas (sysop) 76703,4060
"*S* is for Sarah, who perished of fits; *T* is for Titus, who flew into bits."
—Neil ("*C* is for Cora, who wasted away; *D* is for Desmond, thrown out of a sleigh; *I* is for Ina, who drowned in the lake; *J* is for Jake, who took lye, by mistake.") Rubenking

***There is a reply: 8064

#: 8064
Sb: Dead Child Floating
Fm: Levi Thomas (sysop) 76703,4060
Thanks for the Gorey details. <grin>

***There is a reply: 8076

#: 8076
Sb: Dead Child Floating
Fm: jhon stanley 73765,1026
We called in an agent from M4 to investigate the dead children problem. His name was LEX. He lost his GREP on reality after spending a day YACCing with CAL, the supposed perpetrator. CAL gave him the month but not the DATE. "MORE," said LEX, "TALK, and MAKE my day." Then LEX used his pipe on CAL. He pulled CAL's FINGER out of its socket, which caused LEX's pipe to break.

LEX's partner walked in. He asked LEX to swap space because he had CAL's partner in the other room.

Lisa was a real looker. Her rap sheet was longer than—well, it was long. She was ready to confess. LEX was thinking of other things. Like, was the FTP florist still open?—Lisa needed some roses. "Inode what I was doin'. We took 'em down to the STREAM and let them float away. They was SLEEPing real peacful. What's the DIFF? The superuser would a KILLed 'em when they EXITed anyway."

It was a dark and stormy night. . . .

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 8.**

Object Lesson

It's clear that building all new software from scratch isn't a very productive way to work.

But that's been the only option open for developers. Until now.

Software Evolution

PPI's object-oriented software engineering environment lets designers and programmers build their own reusable Software-ICs®, and use Software-ICs built by others—giving them the same advantages integrated circuits give hardware engineers.

No longer do systems have to be created from scratch.

The PPI environment is the only object-oriented environment available today for C language developers. It includes an integrated family of object-oriented tools that speed up development time, reduce complexity, and simplify future enhancement to systems.

Objective-C® is the object-oriented precompiler that lets you build complex systems from reusable components, using small amounts of new code to attach the parts.

Vici™, the Objective-C interpreter, is a prototyping and debugging tool that lets you test code interactively.

The PPI Software-IC library can be incorporated into programs immediately, as stand-alone components, or as a base for building new components through inheritance.

An Ideal AI Substrate

Objective-C and Vici deliver the kinds of capabilities that AI developers have come to expect from LISP and LISP-derived environments. But since Objective-C and Vici integrate seamlessly with C language, AI applications can now be built to run in the same environment being used for production applications.

Building classes for knowledge representation, inferencing engines and rich user interfaces is natural and straightforward with Objective-C. And integrating knowledge based processing with C applications is easier than ever before.

Real Solutions

Right now, PPI products are in use by many Fortune 500 companies, U.S. Government agencies and major universities. Systems of over 175,000 source lines of code have been completed ahead of schedule and are in commercial use.

In fact, many users report a better than five-fold increase in productivity on their first projects using PPI products.

Software reusability is finally a reality. It's the key to speeding up development time and the only way of achieving an open-ended, future proof system.

To learn how PPI products, training and support can help you meet your development goals faster, while maintaining high quality, call us at (203) 426-1875.

Or write, PPI, Glen Road, Sandy Hook, CT 06482.

"The Concepts of Object-Oriented Programming" is Coming to Your Area.

A two day tutorial that has introduced thousands of professional software developers to this evolutionary technology.

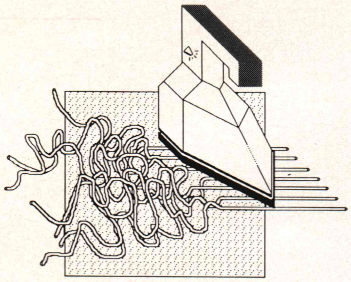
April 23-24 Palo Alto, CA	Sept. 21-22 Chicago, IL
May 18-19 Chicago, IL	Oct. 26-27 Denver, CO
June 11-12 Dallas, TX	Nov. 16-17 Boston, MA
July 20-21 Boston, MA	Dec. 14-15 Palo Alto, CA
Aug. 24-25 Palo Alto, CA	

Call For Details

PPI®

The leader in object-oriented software engineering

THE STATE OF BASIC



The New Face of Subroutines

Those who learned BASIC with popular dialects such as Applesoft BASIC, TRS-80 BASIC, MS-BASIC, and BASICA know how inefficient the *GOSUB <line number>* syntax is. First, the line numbers do not correspond obviously to the tasks that the subroutines carry out. Second, subroutines have no argument lists because these BASIC dialects support global variables only. These conditions make the readability of BASIC programs, especially those lacking numerous comments, difficult. The good news is that the "new wave" BASICs effectively handle these problems. In this issue I will discuss the way in which subroutines are implemented in QuickBASIC and True BASIC, taking advantage of their similar implementations. In the next column, I will look at BetterBASIC's subroutines, which are more like those of Pascal.

The subroutines in QuickBASIC reflect Microsoft's response to the need for more sophisticated syntax. The solution QuickBASIC offers is twofold. First, alphanumeric labels replace line numbers. Thus, you can rewrite an ambiguous *GOSUB 2000* in the old BASIC as *GOSUB READ.FILE*. The label READ.FILE says much more than a line number of 2000!

The second boost to subroutines in QuickBASIC is the implementation of named subroutines that have an optional argument list, much like those in FORTRAN. QuickBASIC supports a strict data interface: all variables used by the subroutine that do not appear in the argument list are local, even if they have identical names to those of variables in the main program. Example 1 (below) shows two simple subroutines: the first clears a screen line, and the other centers text on a specified line number. Subroutine *Center.Text* calls subroutine *ClrLine* and passes the line number specified. The *STATIC* declaration is mandatory and also serves as a reminder that recursive subroutines are not yet implemented.

QuickBASIC subroutine parameters are passed by reference when a variable is used and by value when an expression is used. To protect a scalar variable from being altered by a subroutine, enclose it in parentheses to make it an expression (see commented *CALL ClrLine* in Example 1).

Passing arrays is also simple. QuickBASIC needs to know the number of dimensions the array has when you declare the subroutine. Example 2, page 157, shows a subroutine that calculates the average and standard deviation values of a specified column in a numeric table. The matrix *X* is written as *X(2)* to indicate that it's two-dimensional. QuickBASIC provides the *LBound* and *UBound* functions to return the array's lower and upper bounds, respectively. For one-dimensional arrays you simply enclose the array name in either function to obtain the sought bounds. In the case of multidimensional arrays, a second argument is needed—namely, the dimension number. In Example 2, the *FOR . . . NEXT* loop iterates for all rows in matrix *X*. Assuming that the rows of the numeric matrix are represented by the first dimension and the columns by the second, I use *LBound(X,1)* and *UBound(X,1)* to obtain the row limits. The array-bound functions are very powerful for helping you write general-purpose routines that manipulate arrays of any size.

QuickBASIC also provides the *SHARED* attribute, used with *COMMON, DIM,* and *REDIM* statements, to shorten argument lists of subroutines. The *SHARED* attribute declares the variables and arrays as global and accessible to all routines within a single program. Thus, you should only declare variables that are logically global (that is, needed by most routines) as *SHARED* to avoid the side effects of the old BASICs.

Finally, QuickBASIC subroutines can be exited from using the *END SUB* statement. Subroutines cannot be nested among themselves or with function definitions.

True BASIC implements named subroutines in a similar manner to QuickBASIC. True BASIC does not support labels, so the *GOSUB <label>* syntax is not available—just the *GOSUB <line_number>* (if line numbers are used at all). The preceding discussion of QuickBASIC subroutines applies to True BASIC, with the following exceptions:

- True BASIC supports recursive calls.
- The syntax for declaring arrays in the subroutine argument lists is slightly different. True BASIC requires a comma for each additional dimension. Thus, a simple array *X* is declared as *X( )*, whereas a matrix *X* is written as *X(,)*, and so on. With the advent of True BASIC Version 2, called subroutines passing array arguments

```
SUB ClrLine(Line.Num) STATIC
  LOCATE Line.Num,1
  PRINT STRING$(80, ""); ' clear line
END SUB

SUB Center.Text(T$, Line.Num) STATIC
' Subroutine to center a text
  CALL ClrLine(Line.Num) ' pass Line.Num by reference
  ' or
  ' CALL ClrLine( (Line.Num) ) to pass Line.Num by value
  LOCATE Line.Num, (40 - LEN(T$)/2)
  PRINT T$
END SUB
```

Example 1: *QuickBASIC subroutines to clear a line and center a text on the screen*

can optionally (for enhanced readability) include parentheses, following exactly the same rules as subroutine declarations do.

• True BASIC supports both internal and external subroutines. Internal subroutines are declared within the main BASIC program and before the unique *END* statement. External subroutines can reside in external libraries, modules, or beyond the *END* statement. The difference between the subroutine types is their accessibility to variables in the main program. The main program (up to the *END* statement) is regarded as one programming unit within which all variables are accessible. Thus, internal subroutines can also create and manipulate global variables not appearing in the argument list. Unlike QuickBASIC subroutines, internal True BASIC subroutines have no local variables. This is an important difference to remember if you ever translate programs between the two implementations. External subroutines do not enjoy the same privilege and thus have a stricter data interface, similar to that in QuickBASIC. Examples 3 and 4 (right) show the True BASIC versions of Examples 1 and 2, respectively.

Looking at the subroutines in the new wave BASICs, you can see a touch of FORTRAN present, and why not? They offer a radical solution to a chronic problem that plagued the old microcomputer BASICs. Callable subroutines are also endorsed by Borland International in its new Turbo BASIC. At the time of writing this column, I have Borland's Comdex Fall-86 press release, which indicates that Turbo BASIC will have the more powerful subroutine syntax. Perhaps the Beatles' lyrics from a song on the Sgt. Pepper album provide a suitable comment on BASIC —"It's getting better all the time!"

Vote for your favorite feature/article.
Circle Reader Service **No. 10.**

```
DEF FNMissing = -1E+30 ' define numeric code for missing numbers

SUB Get.Stat(X(2), Col%, Average, StdDev) STATIC
' Get average and stnd. deviation of column Col% of
' two-dimensional array X(,)

 Sum = 0
 SumX = 0
 SumXX = 0

 FOR Row% = LBound(X,1) TO UBound(X,1)
   IF X(Row%,Col%) > FNMissing THEN ' Valid data?
    Sum  = Sum  + 1
    SumX  = SumX  + X(Row%,Col%)
    SumXX = SumXX + X(Row%,Col%)^2
   END IF
 NEXT I


 Average = SumX / Sum
 StdDev = SQR((SumXX - SumX^2/Sum) / (Sum - 1))

END SUB
```

***Example 2:** QuickBASIC subroutine to obtain the average and standard deviation of data stored in an array*

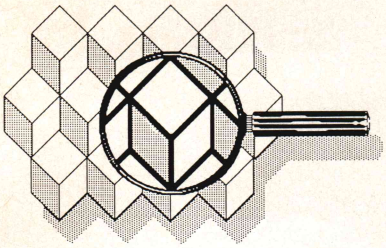
```
SUB ClrLine(Line_Num)
 SET CURSOR 1,Line_Num
 PRINT REPEAT$("" '',80); ! clear line
END SUB

SUB Center_Text(T$, Line_Num)
! Subroutine to center a text
 CALL ClrLine(Line_Num) ! pass Line_Num by reference
 ! or
 ! CALL ClrLine( (Line_Num) ) to pass Line_Num by value
 SET CURSOR (40 - LEN(T$)/2),Line_Num
 PRINT T$
END SUB
```

***Example 3:** True BASIC subroutines to clear a line and center a text on the screen*

```
DEF Missing = -1E+30 ! define numeric code for missing numbers

SUB Get_Stat(X(,), Col, Average, StdDev)
! Get average and stnd. deviation of column Col of
! two-dimensional array X(,)

 LET Sum = 0
 LET SumX = 0
 LET SumXX = 0

 FOR Row = LBound(X,1) TO UBound(X,1)
   IF X(Row,Col) > Missing THEN ! Valid data?
    LET Sum  = Sum  + 1
    LET SumX  = SumX  + X(Row,Col)
    LET SumXX = SumXX + X(Row,Col)^2
   END IF
 NEXT I


 LET Average = SumX / Sum
 LET StdDev = SQR((SumXX - SumX^2/Sum) / (Sum - 1))

END SUB
```

***Example 4:** True BASIC subroutine to obtain the average and standard deviation of data stored in an array*

OF INTEREST



For the Mac

A full-featured AI development system for the Mac, ExperCommon Lisp, from **ExperTelligence**, allows users to develop applications that are independent of the LISP environment. Some of the key features of ExperCommon Lisp include a development environment with more than 1,000 primitives; an incremental compiler that generates 68000 native code directly from LISP source code; a transparent "load-on-call, load-on-return" memory-management/optimization technique; an on-line, automatic symbolic debugger; a class system; and direct access to the Macintosh Toolbox. The debugger, class system, and direct Toolbox access are totally integrated with the compiler. ExperCommon Lisp sells for $995. Reader Service No. 16.
ExperTelligence Inc.
559 San Ysidro Rd.
Santa Barbara, CA 93108
(805) 969-7871

Musicware, a line of products for computer music enthusiasts and recording engineers, is now available from **Opcode Systems**. The line includes two MIDI interfaces that work with the 512K Mac and the Mac Plus. The full-specification Professional Plus MIDI Interface sells for $150, and the Studio Plus MIDI Interface, which features two independent MIDI INs so you can record from two keyboards at once or record and sync at the same time, is priced at $225. Reader Service No. 17.
Opcode Systems
444 Ramona St.
Palo Alto, CA 94301
(415) 321-8977

MagicProducts has introduced the Magic 800K External Disk Drive and the compact MiniMagicDrive for the Mac. The External Drive sells for $199, and the MiniMagicDrive, which is available in 20- or 40-megabyte versions, costs $995 or $1,495. Reader Service No. 18.
MagicProducts Inc.
4505 Spicewood Springs Rd., Ste. 304
Austin, TX 78759
(512) 343-0781

MacC Jr., from **Consulair Corp.**, is an introductory C language development system for the Mac and Mac Plus. A complete K & R implementation of C, MacC Jr. takes full advantage of the Mac's features, letting you produce stand-alone applications in a single step. It includes an extensive standard C and Macintosh support library, Pascal function calls for easy interface to ROM, enhanced symbolic debugging capabilities, and several example programs. MacC Jr. sells for $79.95. Reader Service No. 19.
Consulair Corp.
140 Campo Dr.
Portola Valley, CA 94025
(415) 851-3272

Boards for the IBM PC

The Rocket 286, from **FiveStar Electronics**, is an add-in accelerator board for IBM PCs, PC/XTs, and compatibles. The board provides an 80286 running at 8 MHz and 8K of zero-wait-state cache memory. The hardware switchable 8088 remains in the system for full compatibility with most BIOSs. The Rocket 286 measures 5×9 inches and costs $250. Reader Service No. 20.
FiveStar Electronics Inc.
3220 Commander Dr., Ste. 102
Dallas, TX 75006
(214) 733-4077

Designed for the Compaq 386, IBM PC/AT, PC/XT, and compatibles, Mighty Meg is a new 3.5-megabyte extended memory board from **Quadram Corp.** The board is also upgradable to 14 megabytes with 1-megabit parts and switchless installation. It is designed for use with RAM disks, protected mode operating systems such as ADOS, Topview, and Xenix applications. Mighty Meg is priced from $545 for 0.5 megabyte of memory to $1,475 for 3.5 megabytes. Reader Service No. 21.
Quadram Corp.
One Quad Way
Norcross, GA 30093-2919
(404) 923-6666

PML Systems has released an add-on board and software package that allow non-English speaking PC users to run English-language application programs in their native languages. The PML86 board fits inside a full- or half-length PC expansion slot and comes with a disk containing the foreign-language drivers needed to translate commands into English. Language disks are available for French, Spanish, German, Greek, Italian, Russian, Swedish, Finnish, Thai, Vietnamese, and Sanskrit. With one language disk, PML86 sells for $375. Reader Service No. 22.
PML Systems
3139 E. Almond Ave.
Orange, CA 92669
(714) 771-7744

The Professional Image Board from **ATronics International** allows you to plug a video camera into a PC or compatible and capture live-action images. You can then freeze, computer-enhance, and store pictures on disk. The board sells for $595. Reader Service No. 23.
ATronics International Inc.
1830 McCandless Dr.
Milpitas, CA 95035
(408) 943-6629

Ariel Corp. has introduced a plug-in board that provides a complete signal acquisition, synthesis, and processing system. The DSP-16 combines two channels of high-speed, high-resolution input/output conversion; a large data buffer; and Texas Instruments' second-generation Digital Signal Processing (DSP) microprocessor, the TMS32020. Supplied with the DSP-26 is a software package consisting of the Program Development System and five software application programs: Data Acquisition, Digital Audio Effects, Storage Oscilloscope, Audio Loop Editor, Waveform Synthesizer. The Program Development System

includes all driver routines, a TMS32020 assembler, and debug facilities. A royalty arrangement is available for qualified independent developers. The DSP-16 is priced at $2,495. Reader Service No. 24.

Ariel Corp.
110 Green St., Ste. 404
New York, NY 10012
(212) 925-4155

The CADcard Model 1040 from **Intelligent Graphics Corp.** features an 80186 CPU with 512K of memory dedicated to storing the emulating microcode for IBM's color graphics and professional graphics controller. The CPU provides an additional 380K of storage for display lists, graphic parameters, and user-defined application code. IGC also offers several high-end performance options to the basic unit, including a graphics accelerator, a Z-buffer with hardware hidden surface removal for solids modeling, and an 8087 coprocessor for user application software. CADcard Model 1040 sells for $1,750. Reader Service No. 25.

Intelligent Graphics Corp.
4800 Great Amercia Pkwy.
Santa Clara, CA 95050
(415) 986-8373

Discovery Systems has released an audio-cassette training program for Autodesk's AutoLISP, a training course for AutoCAD users. The eight lessons provide a step-by-step program with complete instructions to create custom AutoLISP functions, custom menus, and other time-saving utilities. The price is $179. Reader Service No. 26.

Discovery Systems
34 Autumnleaf
Irvine, CA 92714
(714) 783-9890

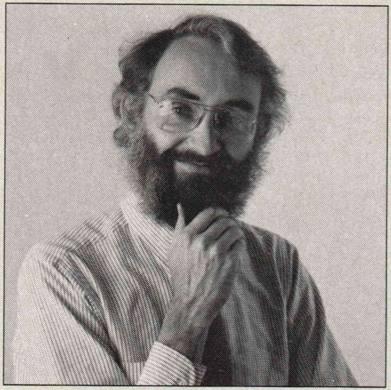
DDJ

ADVERTISER INDEX

***This advertiser prefers to be contacted directly: see ad for phone number.**

ADVERTISING SALES OFFICES

SWAINE'S FLAMES



Mattel and Axlon have been granted permission by the FCC to broadcast robotic control signals to Mattel and Axlon toys as part of a new departure in children's television programming. The toys will move in response to events on screen and shoot at targets on the screen. This scheme is being touted as "interactive," which is nonsense.

There is an interesting precedent for such "interactive" television, and it seems to include all the elements of the present case but one. The precedent had special products that allowed a similar level of "interaction" with the television program; it created two classes of viewers—those who had the products and those who didn't; and it was initially perceived as a departure in children's television programming. What it did not have was an individualized signal sent out over the public airways.

The precedent I have in mind was a cartoon series called "Winky Dink and You" from the days of black-and-white television. If you sent for a special plastic sheet to place over your television screen and special crayons for drawing on it, you could customize W. Dink's on-screen adventures. I don't recall that the producers were accused of economic discrimination, but those were black-and-white times.

Subtract Winky Dink from the Mattel/Axlon scenario and you are left with the individualized television signal and the question of the appropriate use of a limited information channel. I suspect that any effective protest of the plan will focus on that signal. I also suspect that this skirmish, like the Captain Video airwave hijacking of last year, presages an increasing number of battles for bandwidth.

There has been some discussion in our pages and in some more archly academic journals of the failings of conventional implementations of PROLOG as a tool for logic programming. One writer who has not only criticised but suggested ways to bring PROLOG closer to the ideal of logic programming is Lee Naish, who addresses PROLOG's knottiest problems in his book *Negation and Control in Prolog* (Berlin/Heidelberg: Springer-Verlag, 1986).

All PROLOG implementations have some problem with negation. PROLOG is based on Horn-clause logic, and negative information cannot really be expressed in Horn-clause form. PROLOG implementations typically treat negation by more or less identifying it with failure: letting *x cannot be proved* stand for *x is false*. This is often a perfectly reasonable thing to do, but if implemented naively it can lead to illogical conclusions. Naish lays out recommendations for the effective implementation of negation. PROLOG vendors should read them.

The problem with control in PROLOG programs is that semantically innocuous variations in the control logic can cause huge changes in performance. Naish contends that the heuristics of good PROLOG coding that programmers have developed to deal with this problem are by and large simple, effective, and automatable. His approach is to let a preprocessor restructure the code to avoid the worst of the inefficiencies. PROLOG programmers should read this part.

My cousin Corbett has recently been combining catastrophe theory with market analysis and producing surprising results. Catastrophe theory is the fledgling discipline that studies systems on the brink, where the fundamental assumptions on which the study of the systems is built cease to apply.

His inspiration was an article on parallel computers in *High Technology* magazine. Having approached the article to learn about the parallel-computer market, he was nonplussed to find that there was no such thing: that there are degrees of parallelism; that different architectures serve different purposes; that the parallel machines do not compete in one market but define several overlapping, interdependent market fragments.

Expanding his research, Corbett found other computer industry markets in catastrophic transition. The 80386 computer market, with its prenatal cloning, spectre of proprietary designs, and its nonexistent software, has been a particularly fruitful object of study. Then there's the C compiler market, clearly catastrophic, through which Corbett discerns a major fault line developing with optimization on one side and convenience on the other. When the big split comes, Corbett says, half the C market will fall away into ease of use.

The night before it happens, farm animals will be restless.

Oh, yes. If I remember correctly, the Winky Dink television program went away in the face of parental protest. Apparently the producers had not reckoned with the resourcefulness of American youth, who quickly discerned that there was no need to send for the special plastic sheet or the special crayons. You could just draw on the glass with your Crayolas.

Michael Swaine

Michael Swaine
editor-in-chief

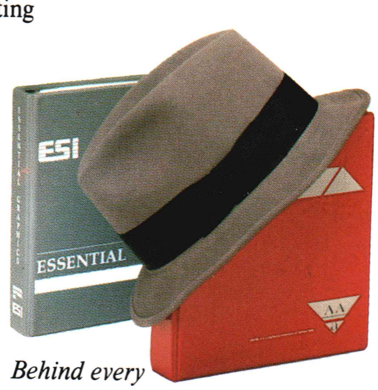
Some Very Impressive People Keep Our Asynch C Tools Under Their Hats

This is the only way we can get some of our customers to take their hats off to us in public. We understand. Most people like to keep a good thing to themselves.

Once you see how powerful and simple our functions are, you'll want to treat our Communications Library Plus as a well guarded trade secret too.

Essential provides the best alternative to Assembly language for communicating via an RS-232.

However, should curiosity get the best of you, call us at 201-762-6965 and we'll let you in on the identity of some of our secret admirers.



Behind every great program is a great library

What Good Are Library Functions If You Can't Get Them To Work?

Providing functions is not enough. Essential Communications Library Plus includes BreakOut, a slick on-line data monitor. BreakOut saves hours of frustration. We know, we used it to debug the XMODEM and other functions in Essential Communications.

No Royalties, 30-Day Guarantee

If within 30 days you don't find our library or BreakOut totally satisfactory, hang the whole thing up and receive a complete refund.

Functions At A Glance

- Interrupt driven to 9600 baud
- Hayes compatible support
- 150 page manual—tutorial
- XMODEM, XON/XOFF support
- Timer/Keyboard functions
- Input buffers to 500K
- Source included
- Demo terminal program
- Demo BBS system
- All major compilers supported
- Ctrl-Break status

Comm Library Plus/BreakOut $250 **Comm Library $185**

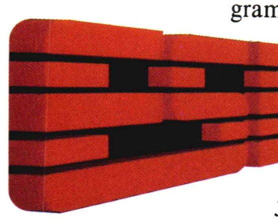
BreakOut On-line Monitor/Debugger

- Monitor RS232 ports up to 9600 baud
- Control comm variables and line signals
- Send/receive data using the scratch-pad editor
- Edit scratch-pad in Hex or ASCII
- Save data to file or re-transmit it
- User configurable keyboard macros
- Use symbols for control chars (<ACK>) for Hex 06

BreakOut $125

Do Your Homework

The library you buy will influence the rest of your programming life. When you've done your homework, you'll choose Essential. Call our support staff of C programmers and find out now how things will be after your check clears.



To order or for support call: 201-762-6965

For foreign orders contact:

England: Gray Matter Tel. (0364) 53499
Japan: Lifeboat Inc. of Japan Tel: 293 4711
West Germany: Omnitex Tel. 07623-61820

Essential Software, Inc.
P.O. Box 1003, Maplewood, New Jersey 07040

NEW

Turbo C®

Turbo C: The fastest, most efficient and easy-to-use C compiler at any price

Compilation speed is more than 7000 lines a minute, which makes anything less than Turbo C an exercise in slow motion. Expect what only Borland delivers: Quality, Speed, Power and Price.

Turbo C: The C compiler for amateurs and professionals

If you're just beginning and you've "kinda wanted to learn C," now's your chance to do it the easy way. Like Turbo Pascal, Turbo C's got everything to get you going.

If you're already programming in C, switching to Turbo C will considerably increase your productivity and help make your programs both smaller and faster. Actually, writing in Turbo C is a highly productive and effective method—and we speak from experience. Eureka: The Solver™ and our new generation of software have been developed using Turbo C.

Turbo C: a complete interactive development environment

Free MicroCalc spreadsheet with source code

Like Turbo Pascal® and Turbo Prolog,™ Turbo C comes with an interactive editor that will show you syntax errors right in your source code. Developing, debugging, and running a Turbo C program is a snap.

Turbo C: The C compiler everybody's been waiting for. Everybody but the competition

Borland's "Quality, Speed, Power and Price" commitment isn't idle corporate chatter. The $99.95 price tag on Turbo C isn't a "typo," it's real. So if you'd like to learn C in a hurry, pick up the phone. If you're already using C, switch to Turbo C and see the difference for yourself.

System requirements

IBM PC, XT, AT or true compatibles. PC-DOS (MS-DOS) 2.0 or later. One floppy drive. 320K.

*Introductory price—good through July 1, 1987

Technical Specifications

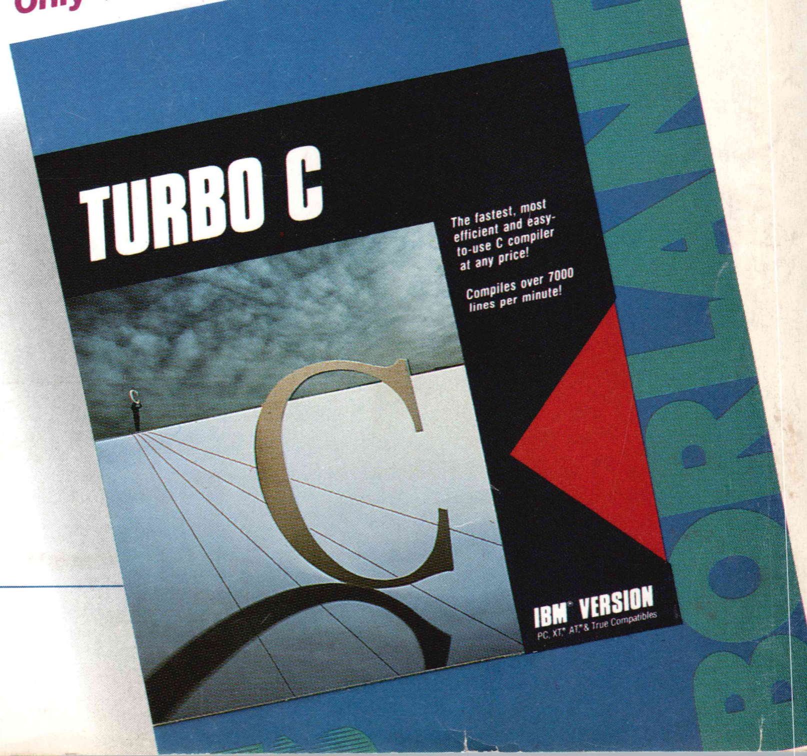
- Compiler: One-pass compiler generating linkable object modules and inline assembler. Included is Borland's high performance "Turbo Linker." The object module is compatible with the PC-DOS linker. Supports tiny, small, compact, medium, large, and huge memory model libraries. Can mix models with near and far pointers. Includes floating point emulator (utilizes 8087/80287 if installed).
- Interactive Editor: The system includes a powerful, interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code.
- Development Environment: A powerful "Make" is included so that managing Turbo C program development is highly efficient. Also includes pull-down menus and windows.
- Links with relocatable object modules created using Borland's Turbo Prolog into a single program.
- ANSI C compatible.
- Start-up routine source code included.
- Both command line and integrated environment versions included.

Sieve benchmark (25 iterations)

	Turbo C	Microsoft® C	Lattice C
Compile time	**3.89**	16.37	13.90
Compile and link time	**9.94**	29.06	27.79
Execution time	**5.77**	9.51	13.79
Object code size	**274**	297	301
Price	**$99.95**	$450.00	$500.00

Benchmark run on a 6 Mhz IBM AT using Turbo C version 1.0 and the Turbo Linker version 1.0; Microsoft C version 4.0 and the MS overlay linker version 3.51; Lattice C version 3.1 and the MS object linker version 3.05.

Only $99.95!*



BORLAND
INTERNATIONAL
Vive la différence

4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

TC 11

For the dealer nearest you or to order by phone call

(800)255-8008

in CA (800) 742-1133 in Canada (800) 237-1136

Circle **no. 161** on reader service card.